Revista Digital

ATIX



atix Libre
Fundación

Equipo Editorial

Dirección y Coordinación General

Esteban Saavedra López (esteban.saavedra@atixlibre.org)

Diseño y Maquetación

Jenny Saavedra López (jenny.saavedra@atixlibre.org)

Esteban Saavedra López (esteban.saavedra@atixlibre.org)

Revisiones

Esteban Saavedra López

Jenny Saavedra López

Noticias

Jenny Saavedra López

Autores Frecuentes

Esteban Saavedra López

Martín Márquez

Rafael Rendón

Ernesto Rico Smith

Herramientas

La edición de esta revista fue realizada de forma integra haciendo uso de Software Libre

atix

El que lo Intenta

El que lo Sabe

El que lo Puede

El que lo Logra

Editorial

Trabajo y Tradición

Como nos habíamos fijado desde el inicio de este proyecto, el brindar un trabajo contínuo y que contribuya al saber de nuestros imnumerables lectores, fortaleciendo sus conocimientos y destrezas en el amplio mundo del software, conocimiento, tecnologías y cultura libre, presentamos esta nueva entrega.

Con el título de este número deseamos expresar dos palabras de gran significado, el TRABAJO que representa el esfuerzo que todas las personas realizan en alguna actividad en particular y la TRADICIÓN que para muchos representa una limitante o un nivel de conformismo, sin embargo en un sentido más amplio la TRADICIÓN representa mantener nuestros valores, creencias, costumbres y convicciones, para ser transmitidos a las nuevas generaciones, sin que esto signifique aislarnos ni descuidar los continuos cambios que se suscitan y son motivos de imnovación.

En este segundo número de este año, continuamos mostrando las diferentes alternativas de desarrollo de aplicaciones orientadas a la web, el trabajo con datos especializados en diversos ambientes y plataformas, el uso de herramientas sencillas para automatizar trabajos de administración y desarrollo y por último el avance en el desarrollo de un excelente trabajo en la gestión de infraestructuras de red.

Bienvenidos a nuestro vigésimo primer número



Esteban Saavedra López
Presidente Fundación AtixLibre

Liberado el 18 de febrero del 2013

{CONTENIDO}

Introducción a ZK y su relacion con RIA (Parte 1)

ZK es un framework de aplicaciones web en AJAX, software de código abierto que permite una completa interfaz de usuario para aplicaciones web

¿Qué es ZK?

- ZK es un proyecto libre creado por la empresa Potix que nació con el objetivo de simplificar radicalmente el desarrollo de aplicaciones web.
- ZK es AJAX sin escribir JavaScript.
- ZK es un framework de componentes dirigido a través de eventos (Event-Driven). Con él podemos desarrollar interfaces de usuarios de un modo profesional y extremadamente fácil.
- Open Source, pero además detrás tiene el respaldo de una compañía POTIX. Más adelante veremos los diferentes tipos de paquetes que podemos descargar.
- Está basado en tecnologías abiertas, con una curva de aprendizaje casi plana:
  - XHTML (HTML escrito con la sintaxis de XML)
  - XUL (**http://www.mozilla.org/projects/xul/**)
- Funciona también con JSP, JSF, Portlet, tecnologías Java EE y se integra con los IDE's más comunes. En el caso de Eclipse por ejemplo con ZK Studio.
- Diseñado para ser Direct RIA (Direct Rich Internet Applications).

¿Quién lo usa?

- Oracle, eBay, Samsung, Barclays, Toyota, etc.

Tenéis un reporte fantástico sobre quíen usa ZK en: **http://www.zkoss.org/whyzk/WhosUsing**

¿Dónde lo encuentro?

ZK está disponible para ser descargado en **www.ZKoss.org** en varias modalidades de licencia **http://www.zkoss.org/license/**.

- Community Edition, completamente libre para su uso en OpenSource y Particular
  - ZK CE - Licencia LGPL
- Para profesionales
  - ZK PE - Licencia ZOL (ZK OpenSource Licence) o Licencia Comercial
- Para empresas u organizaciones
  - ZK EE - Licencia ZOL (ZK OpenSource Licence) o Licencia Comercial

Y las diferencias entre las licencias expuestas

http://www.zkoss.org/whyzk/Features



Características principales

- Diseñado para ser extremadamente ligero:
  - Sin dependencias
  - No requiere plugins de ningún tipo.
- Responsive Design
- Responsive Components
- Compatible con la mayoría de navegadores existentes, incluso legacy (y sin sorpresas):



Figura 1. Navegadores

- ZK también soporta los navegadores de dispositivos móviles, de hecho existe ZK Mobile, que es ZK aplicado al desarrollo de aplicaciones Móviles, accesibles por el navegador de los mismos.
- Se comporta de igual modo en todos los navegadores
- Se renderiza lo mismo para el usuario, es independiente del decorador que utilice el navegador según el sistema operativo.
  - Por ejemplo pintando un botón en Mac, Windows o Linux.



Figura 2. Botones de sistema

  - La decoración de los componentes no depende del sistema operativo. Y es completamente personalizable.
- Los componentes que forman ZK son una representación POJO (Plain Old Java Objects) de todos los componentes XHTML y una batería adicional de todos los componentes del propio ZK. En total unos 200.

Otras características importantes

- **100% Basado en componentes:**
  - Para el programador, todos los componentes de la interfaz de usuario son POJOS, y son completamente operables desde el API de Java.
  - Los componentes tienen atributos, 0 o n.
  - Los componentes tienen 0 o n eventos, que son ejecutados según el usuario interactua.
- **Seguridad**:
  - No se expone la lógica de negocio al cliente, o información a internet.
- **Documentación**:
  - La documentación es abundante y muy actualizada para las diferentes versiones de los productos de terceros.
- **Avanzado**:
  - ZK selecciona permite configurar el Server Push (basade en Comet) de forma transparente, a su vez escoge la estrategia e implementación del mismo automáticamente.
  - ZK permite mediante CSA (Client side actions) ejecutar eventos en el cliente. Puesto que no tiene sentido ejecutar un rollover o animación con Ajax. Aún así, CSA permite escuchar en el servidor

los eventos (onfocus, onblur, onmouseover...) para trabajar con ellos.

- **Extensibilidad**:
  - ZK permite crear componentes desde 0 o extenderlos, incluso conjuntos de ellos de varias formas. Directamente en un fichero ZUL, dentro de el mismo, o desde Java.
  - ZK Mobile aporta desarrollo para aplicaciones online via Browser.
  - ZK Spring, integra ZK con Spring MVC, Spring Web Flow y Spring Security.
  - ZK JSP Tags y ZK JSF Components hace posible enriquecer aplicaciones legacy con ZK.
  - ZK abstrae de los problemas de compatibilidad entre navegadores, incluso de IE 6.
  - ZK Richlets para crear mini-aplicaciones integrables en webs hechas en cualquier tenología.
  - ZK con Liferay, con Jboss Seam, JasperReports... etc.
  - ZK JSR 299 CDI :) `http://blog.zkoss.org/index.php/2010/01/07/integrate-zk-and-jsr-299weld/`
- **Accesibilidad**:
  - ZK Accesibility - `http://www.zkoss.org/zk508/`
  - Niveles de conformidad - `http://www.zkoss.org/zk508/levelsOfConformance.htm`
  - Artículos "How to Make Your AJAX Applications Accessible" - `http://www.zkoss.org/zk508/additionalArticles.htm`

Dónde y Cómo utilizar ZK

- Es una plataforma perfecta para montar prototipos y probar código.
- Es completamente factible utilizarlo en entornos altamente explotados por los usuarios.
- Podemos crear simples Richlets web, que son componentes con todo lo necesario para funcionar dentro de otras páginas hechas en cualquier tecnología, respondiendo a una simple url.
- Es una tecnología completamente madura, que existe como tal desde el año 2005 y ha tenido una comunidad que no ha parado de crecer de una forma increíble.
- Dispone de una empresa por detrás que reespalda y coordina todo su desarrollo.

Dónde no utilizar ZK

Puesto que ZK utiliza los eventos que el navegador y el servidor web generan, al igual que cualquier framework RIA, no sirve para para aplicaciones del tipo:

- Videojuegos de acción.
- Aplicaciones basadas en gráficos vectoriales o tridimensionales
- Programas de edición fotográfica o de video

Acerca de este documento

Este documento es un extracto de la documentación oficial del Framework ZK, traducido y ampliado por Francisco Ferri. Colaborador de Potix (creadores del Framework ZK). Si quieres contactar con él puedes hacerlo en `franferri@gmail.com`, en Twitter `@franciscoferri` o en LinkedIn: `http://www.linkedin.com/in/franciscoferri`

Referencias

[1] http://www.zkoss.org/

[2] http://books.zkoss.org/wiki/ZK_Installation_Guide/Quick_Start/Create_and_Run_Your_First_ZK_Application_Manually

Autor



Francisco Ferri
Colaborador de Potix (ZK Framework)
Jefe de Proyecto Freelance en ISBAN (Banco Santander)
Consultor Freelance
Twitter: @franciscoferri
franferri@gmail.com
franferri@javahispano.org

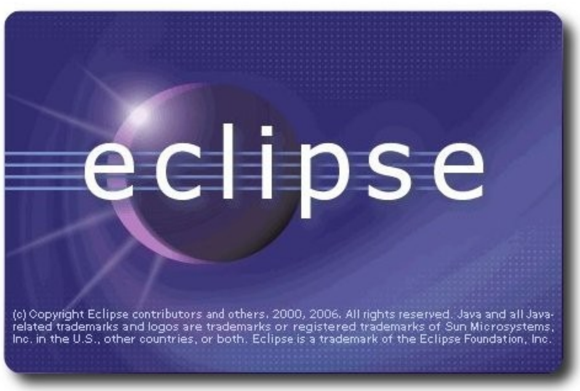


Introducción a ZK y su relacion con RIA (Parte 2)

ZK es un framework de aplicaciones web en AJAX, software de código abierto que permite una completa interfaz de usuario para aplicaciones web

En esta segunda parte veremos la configuración del entorno de desarrollo, tomando en cuenta las herramientas y configuraciones necesarias para disponer de un entorno que facilite el desarrollo de aplicaciones en ZK.

Configurar Eclipse



Para la realización de este tutorial usaremos Eclipse 3.7 Indigo, la distribución para Java EE.

Podemos descargarlo de:

`http://www.eclipse.org/downloads/packages/release/indigo/sr2`.

Una vez descargado, extraemos el contenido del **`zip/tar.gz`** en una carpeta y lo ejecutamos (**`eclipse.exe`** o **`./eclipse`**).

Las páginas de interfaz de usuario en ZK son ficheros xml que utilizan la extensión "zul".

Para editar páginas de interfaz de usuario de ZK en Eclipse, añadimos “zul” a la lista de páginas de tipo XML dentro de las preferencias de Eclipse, para lo cual procedemos del siguiente modo:

1. Seleccionamos el menú superior del **`Eclipse Window / Preferences`** para abrir la ventana de preferencias.
2. En el panel de la izquierda, seleccionamos General y dentro de ella Content Types como muestra la imágen.
3. En el panel de la derecha expandimos el nodo Text en la lista de Content types y finalmente seleccionamos XML.
4. Hacemos clic en el botón Add y escribimos **`*.zul`**, y finalmente pulsamos el botón de OK.

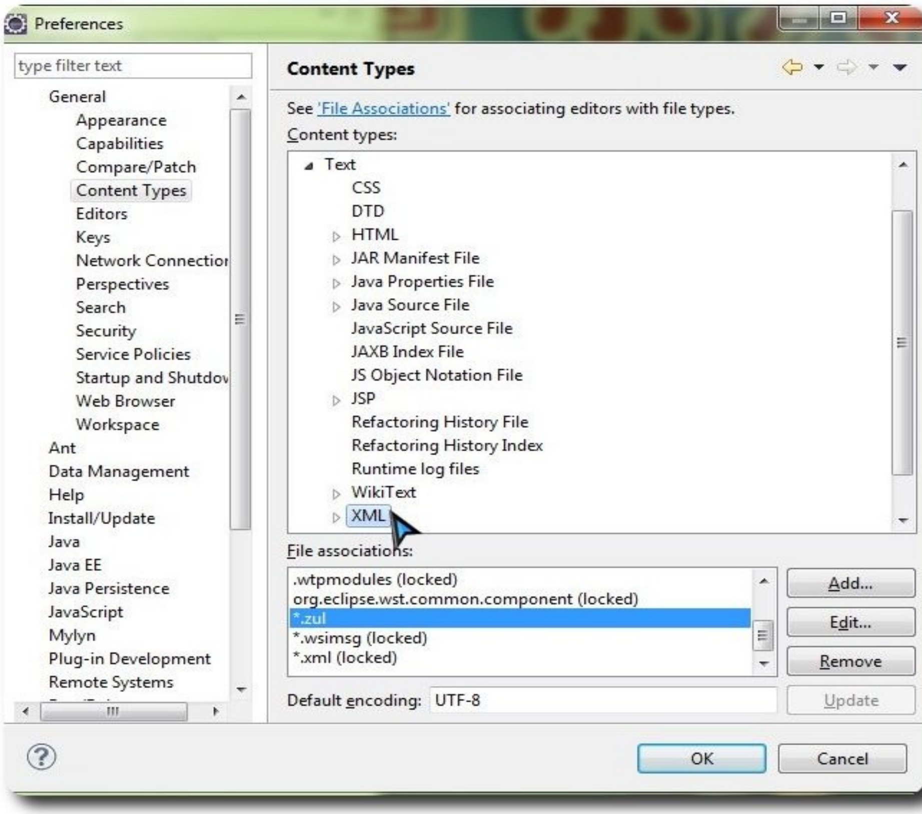


Figura 1.

Una vez hayamos hecho esto, Eclipse usará el editor XML para abrir los ficheros ZUL, y los reconocerá como tal.

Instalar ZK en un Proyecto Web de Eclipse

Descargamos ZK

Primero debemos descargar ZK (Community Edition por ejemplo), puedes hacerlo directamente del sitio oficial: **`http://www.zkoss.org/download/zk`**.

Una vez descargado, reconocerás el fichero porque llevará por nombre algo parecido a **`zk-bin-[version].zip`**. Extraer su contenido en una carpeta.

Creamos un Proyecto en Eclipse

Para crear una aplicación web en Eclipse, primero, creamos un Dynamic Web Project:

1. Seleccionamos `File / New / Dynamic Web Project`
2. Escribimos como nombre del proyecto, por ejemplo warmup, o prueba, y dejamos el resto de configuraciones como están por defecto.

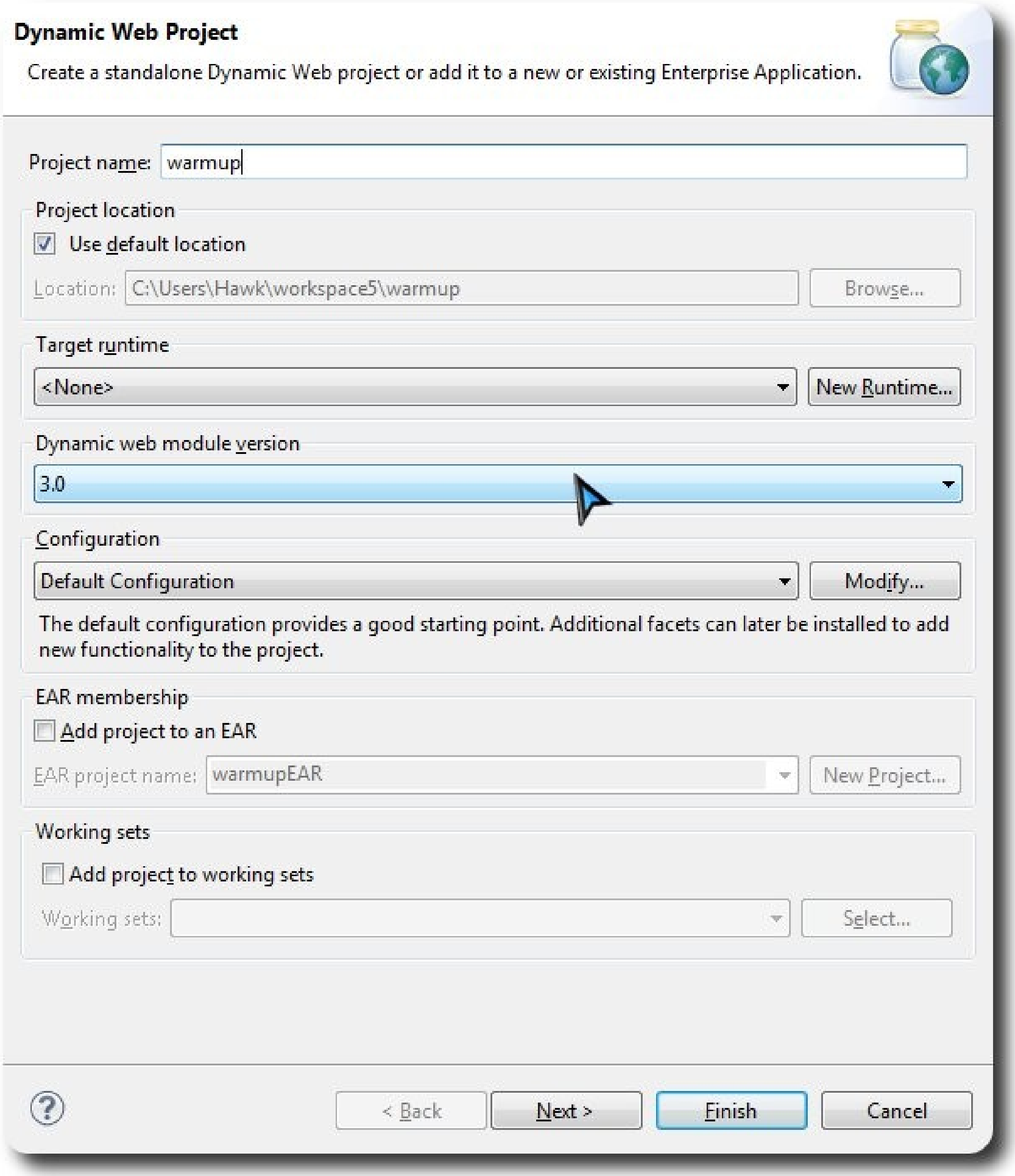


Figura 2.

- ✔ Puedes dejar configurada la opción Target runtime como None
- ✔ Fíjate en la imagen que usamos Dynamic web module version, la versión 3.0. El motivo es que usando Servlet 3.0 aplicación web requiere menos configuración para funcionar.

✔ Si prefieres usar un servidor de aplicaciones web que soporte versiones anteriores a la especificación Servlet 3.0 o la JDK 1.5, tienes que añadir más configuración en el fichero **`web.xml`**. Dispones de más información en la Guía de Instalación Manual de ZK y ejemplos de **`web.xml`**, al final del artículo, en la sección de referencias.

Instalar el JAR de ZK en el Proyecto Web

Para usar ZK en un proyecto, tienes que copiar el JAR de ZK en tu carpeta de librerías de la aplicación (library).

Los ficheros JAR de ZK están en la carpeta donde hemos extraído el zip de ZK, concretamente dentro de las subcarpetas **`WebContent/WEB-INF/lib`**.

```
{YOUR_ZK_UNZIP_FOLDER}\dist\lib
{YOUR_ZK_UNZIP_FOLDER}\dist\lib\ext
```

Es decir, uno a uno, todos los ficheros .jar que estén en esas carpetas, los copiamos a nuestra carpeta de librerías del proyecto web.

Crear una página simple

Después de la instalación, puedes crear una simple página zul para verificar si ZK está funcionando o no.

Desde el eclipse:

1. Para añadir un nuevo fichero seleccionamos **`File / New / File`**, o también **`File / New / Other / File`**. Lo llamaremos hello.zul y lo colocaremos en la carpeta WebContent de nuestro proyecto Web.
2. Hacemos doble clic para editarlo y nos vamos a la pestaña de Source, para poder editar su contenido como texto.

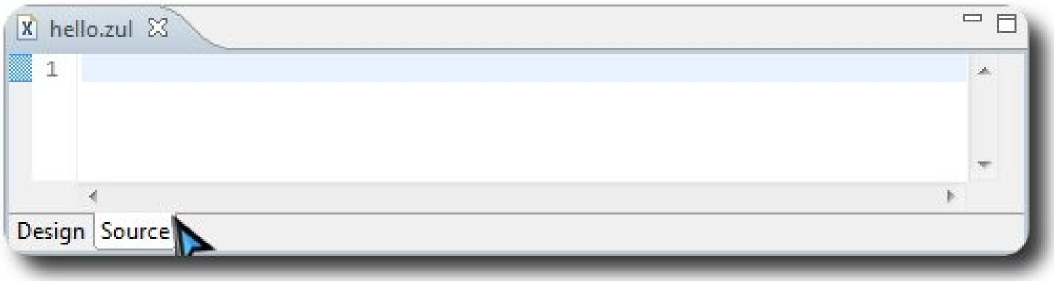


Figura 3.

3. Copiamos y pegamos el siguiente código fuente de ejemplo dentro del fichero (hello.zul) y lo guardamos.

```
<window title="My First ZK Application" border="normal">
   Hello World!
</window>
```

Ahora, en la vista Project Explorer de Eclipse, nuestro proyecto será similar a:

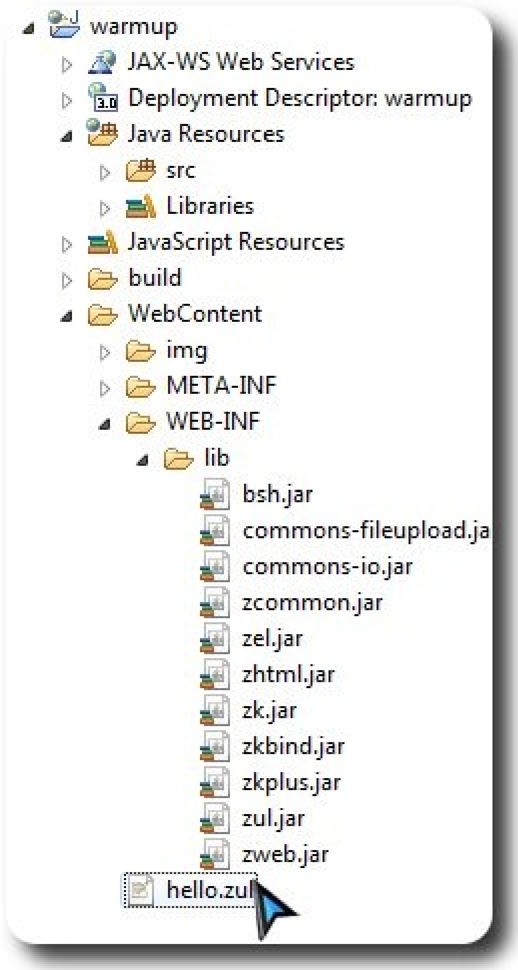


Figura 4.

Si no puedes encontrar en Eclipse la vista Project Explorer seleccionala en el menú **`Window / Show View / Project Explorer`** para abrirla.

Configuramos el servidor

Antes de ejecutar una aplicación web, tenemos que configurar un servidor en Eclipse. Para ello seleccionamos **`Window / Preferences`**, y en la parte izquierda de la ventana de preferencias que nos ha aparecido seleccionamos Server / Runtime Environments. Pulsamos Add para añadir una configuración de ejecución de servidor.

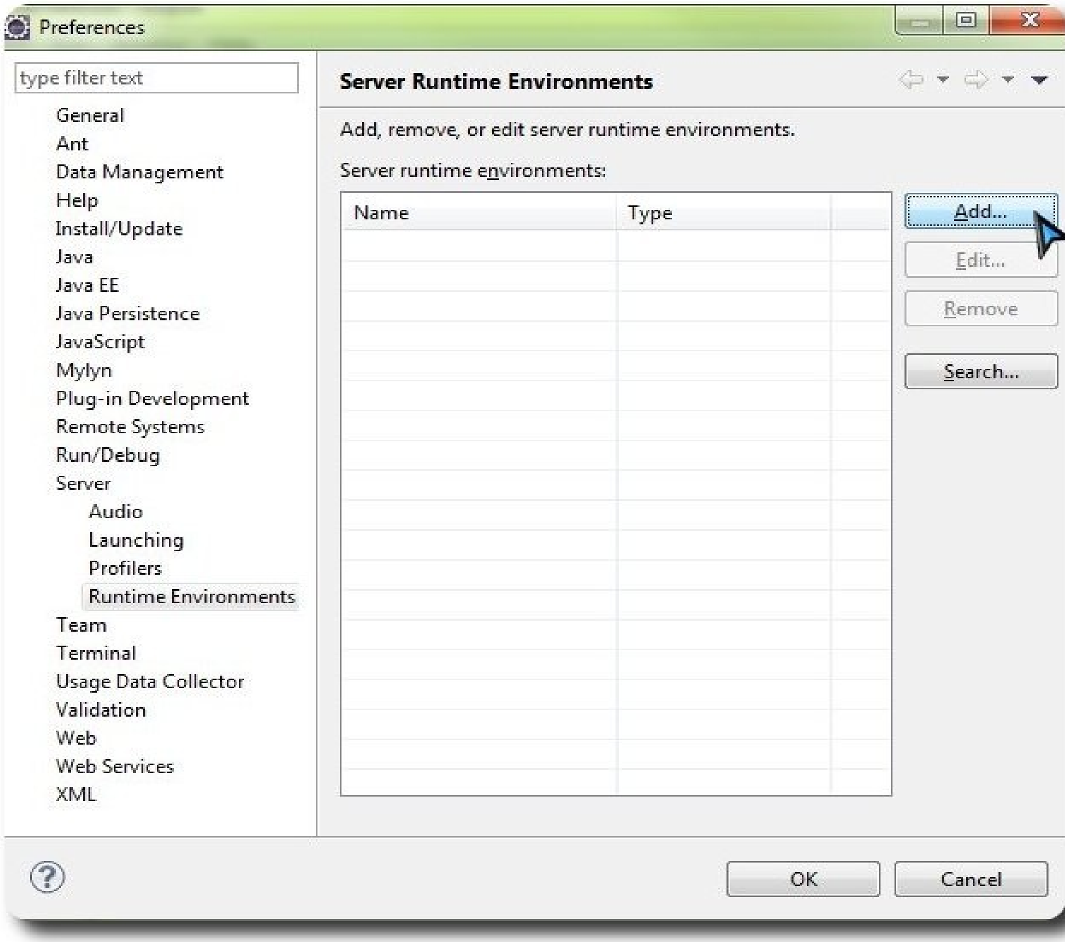


Figura 5.

Seleccionamos **`Apache / Tomcat v7.0 Server`**, puesto que soporta Servlet 3.0, y marcamos Create a new local server, a continuación hacemos clic en siguiente.

Si utilizas JDK 1.5, puedes elegir Tomcat v6.0, pero necesitarás configurar manualmente algunos aspectos en el fichero **`web.xml`**. Dispones de más información en la Guía de Instalación de ZK.

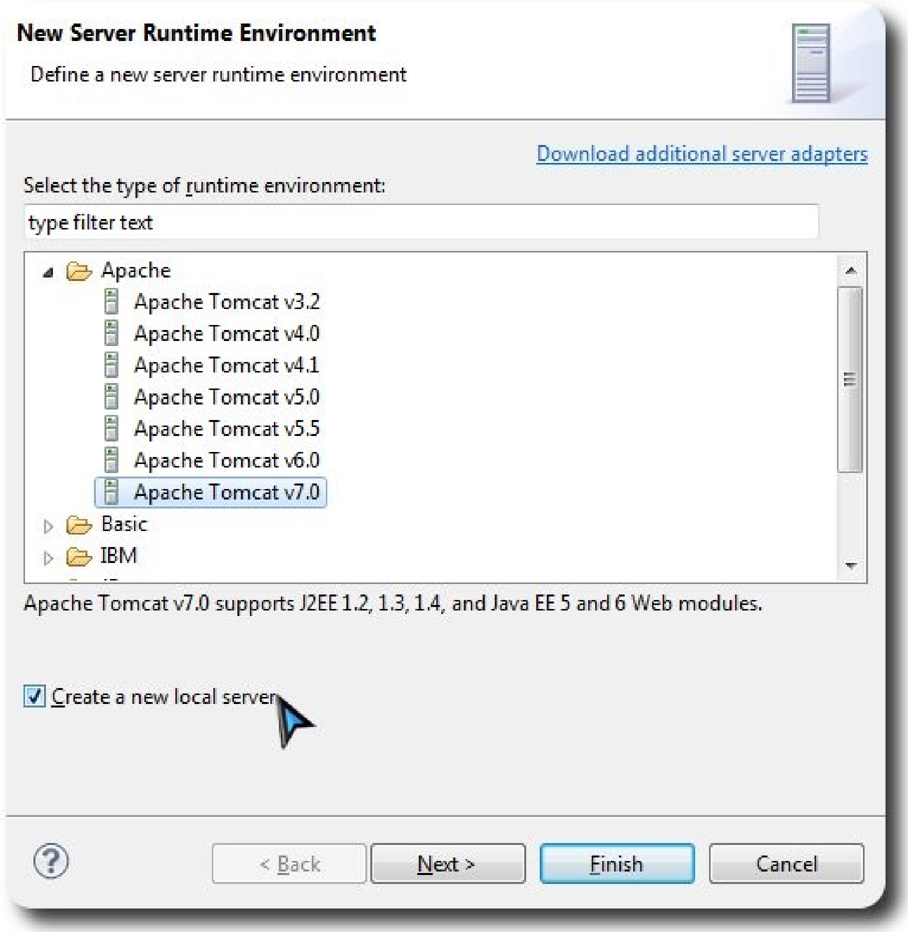


Figura 6.

Si has instalado Tomcat 7 anteriormente, simplemente indica el directorio de instalación en "**`Tomcat installation directory`**".

Si no tienes instalado Tomcat 7 puedes descargarlo siguiendo estos pasos:

1. Haz clic en Download and Install y elige la carpeta de destino.

   La ruta del directorio de destino no debe contener carácteres que no sean ASCII (como por ejemplo acentos).

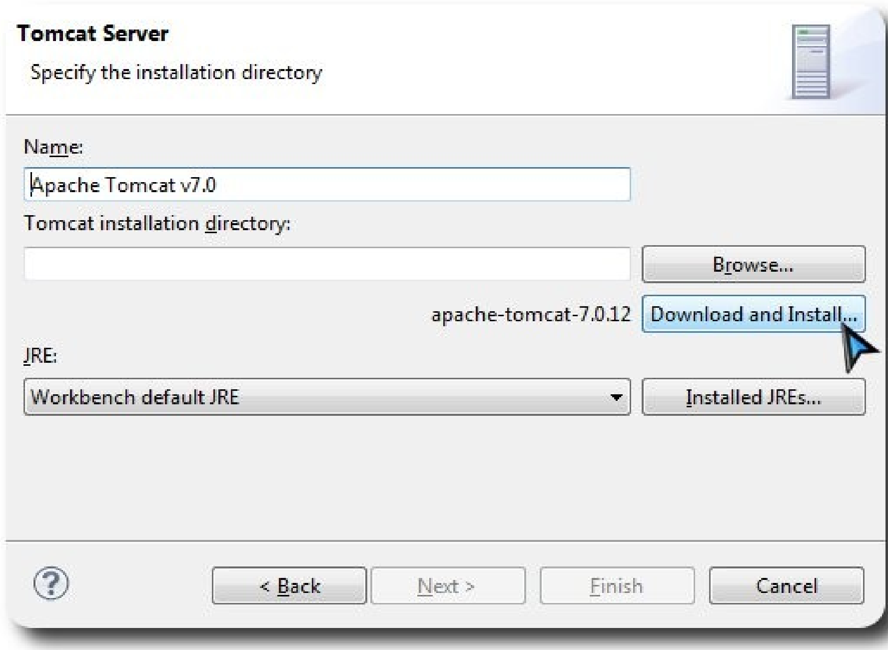


Figura 7.

2. Acepta la licencia y espera.

   Eclipse te mostrará un mensaje de error antes de que la instalación termine, simplemente ignoralo.

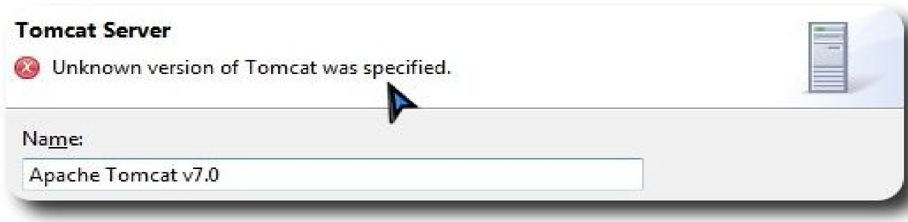


Figura 8.

   Para garantizar que la instalación se realiza correctamente, no pares la descarga ni interrumpas la actividad del Eclipse hasta que termine.



Figura 9.

   Eclipse terminará de descargar e instalar el Tomcat en la carpeta especificada.

3. Cuando todo el proceso termine, pulsa Finish.

   A partir de este momento podrás ver la nueva entrada en `Server runtime environments` en la pantalla de `Preferences`. Pulsamos OK.

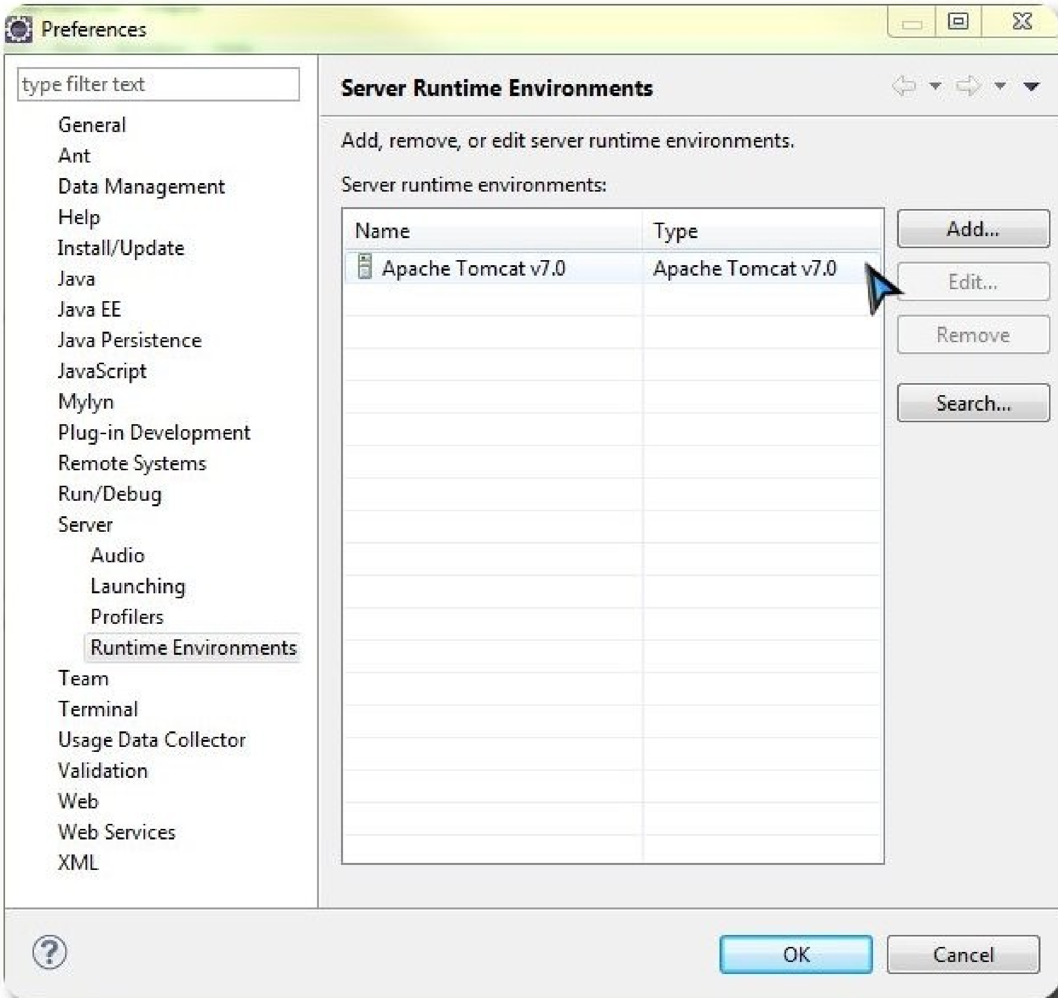


Figura 10.

Ejecutamos la aplicación

Ahora hacemos clic con el botón secundario del ratón sobre el fichero "`hello.zul`" y seleccionamos **`Run As / Run on Server`** para ejecutar el fichero en nuestro servidor de aplicaciones.

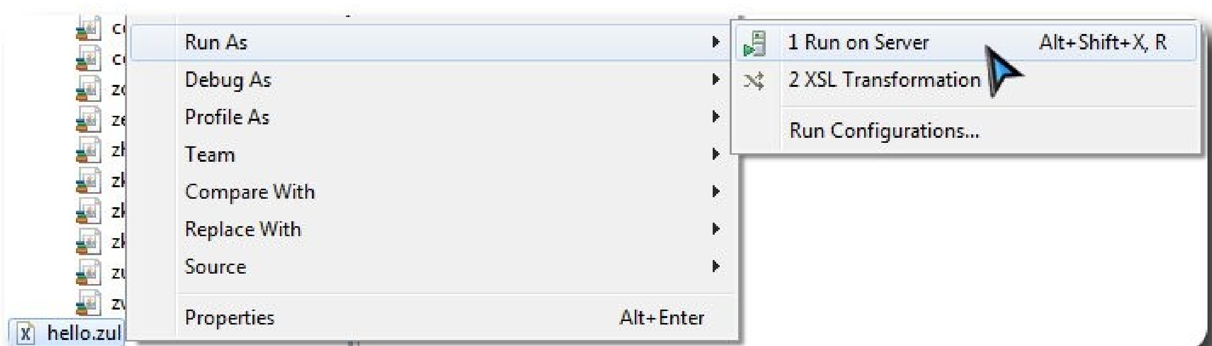


Figura 11.

Elegimos el servidor Tomcat v7.0 que nos aparece. Y podemos marcar la opción Always use this server when running this project para evitar que nos pregunte lo mismo cada vez que ejecutemos el fichero en el servidor de aplicaciones.

Hacemos clic en Finish y esperamos a que el servidor arranque.

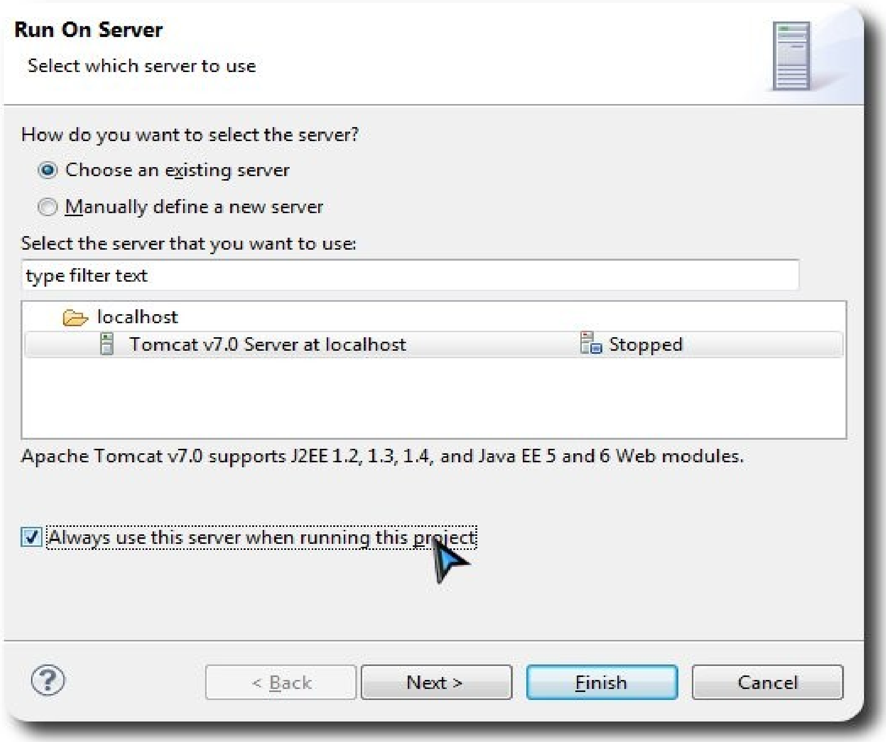


Figura 12.

Después de que el servidor arranque, Eclipse abrirá su navegador y conectará con el servidor de aplicaciones automáticamente: **`http://localhost:8080/hello.zul`**. Si lo que ves es muy similar a la siguiente imagen significa que tienes tu proyecto listo para usar ZK:

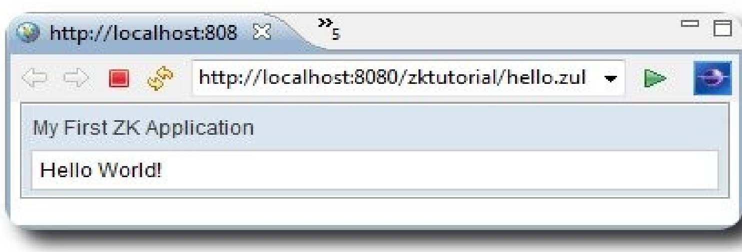


Figura 13.

Puedes volver y seguir estos mismos pasos para ejecutar cualquier fichero **`.zul`** de tu proyecto.

Acerca de este documento

Este documento es un extracto de la documentación oficial del Framework ZK, traducido y ampliado por Francisco Ferri. Colaborador de Potix (creadores del Framework ZK). Si quieres contactar con él puedes hacerlo en **`franferri@gmail.com`**, en Twitter **`@franciscoferri`** o en LinkedIn: **`http://www.linkedin.com/in/franciscoferri`**

Referencias

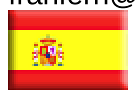
[1] http://www.zkoss.org/

[2] http://books.zkoss.org/wiki/ZK_Installation_Guide/Quick_Start/Create_and_Run_Your_First_ZK_Application_Manually

Autor



Francisco Ferri
Colaborador de Potix (ZK Framework)
Jefe de Proyecto Freelance en ISBAN (Banco Santander)
Consultor Freelance
Twitter: @franciscoferri
franferri@gmail.com
franferri@javahispano.org



Play Web Framework (Parte 2)

En el mundo de desarrollo de software, el uso de los frameworks se ha convertido en una herramienta poderosa, eficiente y eficaz al momento de afrontar proyectos de desarrollo de software, por su fácil aprendizaje, rápido desarrollo y estructura robusta.

Consecuente a la primera parte sobre Play! Framework, continuamos con otro tema importante e indispensable dentro del desarrollo web.

Web 2.0

Hace un par de años el término web 2.0 ha tomado importancia debido a la evolución de las redes, tanto en ámbitos de infraestructura como de tecnología. Tales cambios implican nuevas técnicas y posibilidades para el mundo de desarrollo.

En el caso de web 2.0 el más claro ejemplo son las redes sociales, siendo las más populares aquellas que brindan dinamismo y fluidez visual a los usuarios finales. Estas características ahora son tan comunes que no ofrecerlas afecta la ubicación y popularidad de mercado de cualquier tipo de aplicación web.

Específicamente podemos hablar de estos temas en lo que se refiere a la web 2.0:

- ✔ Hojas de estilo (CSS)
- ✔ REST
- ✔ Comportamiento dinámico (Javascript)

Estos elementos no son nuevos en el ámbito del desarrollo web, pero su estandarización y crecimiento en los diferentes motores de navegación (e.g. javascript V8 en chrome) ha creado un momento idóneo para el desarrollo de aplicaciones web dinámicas.

Contenido estático

En el artículo anterior vimos un ejemplo de aplicación web simple y completo. Su comportamiento estaba completamente automatizado a través del plugin CRUD agregado a Play!.

Para aquellos casos en que se requiera definir las pantallas de interfaz gráfica de forma personalizada, se puede usar el motor de plantillas que Play! ofrece, su arquitectura es la siguiente:

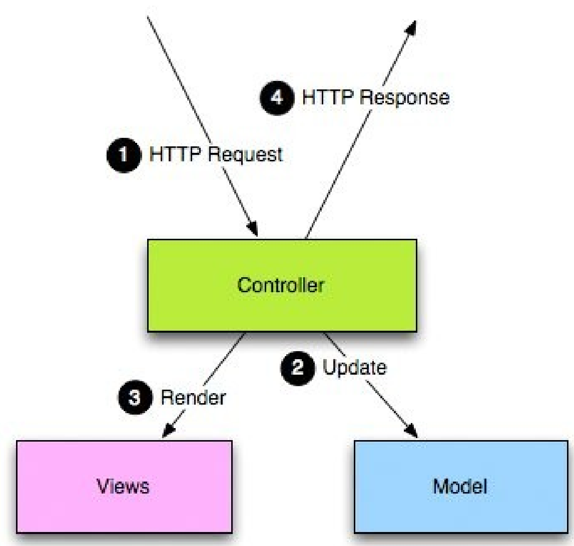


Figura 1. Ciclo de via de una pagina en Play!

En el esquema anterior visualizamos los componentes View o Vistas, las cuales

definen lo que el usuario final puede ver y hacer a través de controles y comandos HTTP. Play! toma todas las acciones ejecutadas en las vistas y las mapeas a través de rutas normalizadas a un controlador específico, el cual define los resultados y operaciones a seguir.

El motor de plantillas

Play! utiliza un motor de plantillas que permite generar contenido basado en texto (HTML, XML, JSON) usando el lenguaje interpretado Groovy como base, y también provee un sistema de tags que permiten crear funciones reutilizables.

Su sintaxis se basa en los elementos:

Expresiones: Sentencias escritas en Groovy que permiten interacciones entre el código Java estático desde las plantillas HTML o JSON.

Exploración (prefijo = $)

Sentencias que permiten acceder a atributos y operaciones de componentes Java, permite llamar a operaciones y realizar acciones condicionales.

Por ejemplo, para mostrar el nombre de una tarea presente en la base de datos usamos:

```
<h1>Tarea ${tarea.nombre}</h1>
```

Acción (prefijo = @, @@)

Permite realizar llamadas a operaciones estáticas de los controladores, es decir, permite interactuar directamente con el código estático en Java.

Por ejemplo, para generar una URL de redirección hacia la acción que muestra todas las tareas usamos:

```
<a
href="Tareas.showTareas(tarea.id)">Todas
las tareas</a>
```

De la misma forma, podemos relacionar eventos HTML con llamadas a código en los controladores.

i18n (prefijo = &)

Play! permite definir mensajes para diferentes idiomas, de forma que la aplicación puede tener mensajes tanto en español como en inglés en nuestra interfaz de usuario. Para tal efecto se definen mensajes en idiomas específicos en los archivos dentro de 'conf/messages' para el idioma por defecto y en '`conf/messages.en`' para inglés.

El archivo '`conf/messages`' tiene su contenido de la siguiente forma:

```
taskName = El nombre de la tarea es %s
```

En '`conf/messages.en`' sería (la %s representa las cadenas pasadas como parámetros):

```
taskName = Task's name is : %s
```

Para acceder a estos valores en las plantillas usamos:

```
<h1>&{taskName, tarea.nombre}<h1>
```

Tags

Los Tags son fragmentos de plantillas que pueden ser invocados usando parámetros para modificar su comportamiento. Por ejemplo, para agregar un script a nuestra página usamos:

```
#{script 'jquery.js' /}
```

O para iterar una colección de elementos, podemos usar una sintaxis '`foreach`':

```
<h1>Client ${client.name}</h1>
<ul>
#{list items:client.accounts,
as:'account' }
<li>${account}</li>
#{/list}
</ul>
```

Decoradores

Elementos que permiten una alternativa simple para definir y compartir diseño y estilos de página entre plantillas. Por ejemplo, para extender una página base se

puede usar:

```
#{extends 'simpledesign.html' /}
#{set title:'A decorated page' /}
This content will be decorated.
```

Donde todos los estilos y diseños definidos en 'simpledesign.html' serán usados para decorar nuestra página.

Todos estos elementos se usan y definen en los archivos dentro de '**app/views**'.

Ejemplo

Crearemos un diálogo de agregación y listado simple de Tareas que nos permitan filtrarlas en base a su nombre. En este caso no usaremos el plugin CRUD sino que veremos todo el ciclo de vida completo.

Primeramente creamos la estructura de un nuevo proyecto con:

```
$> play new Tareas
$> cd Tareas
```

Creamos el archivo '**app/models/Tarea.java**' con el siguiente contenido:

```
package models;

import play.*;
import play.db.jpa.*;
import play.data.validation.*;

import javax.persistence.*;
import java.util.*;

@Entity
public class Tarea extends Model{

	@Required
	public String titulo;
	public boolean completada;

	public Tarea(String titulo){
		this.titulo = titulo;
	}

	public static List<Tarea> findByTitulo(String titulo){
		return Tarea.
		find( "SELECT t FROM Tarea t WHERE lower(t.titulo) like :titulo" )
		.bind("titulo","%" + titulo.toLowerCase() + "%")
		.fetch();
	}
}
```

Esta es nuestra entidad persistente Tarea, la cual define un título y su estado. Adicionalmente agrega una operación '**findByTitulo**' que filtra las tareas que tengan títulos similares a la cadena recibida como parámetro.

Configuramos una base de datos en memoria modificando el archivo '**conf/application.conf**' con: **db = mem**

Esta configuración define una base de datos en memoria, la cual nos permite trabajar de

forma local sin necesidad de un sistema de base de datos externo, sus datos son perdidos al reiniciarse la aplicación.

Creamos un controlador para nuestra nueva página en '**`apps/controllers/Tareas.java`**':

```
package controllers;

import play.*;
import play.mvc.*;
import play.data.validation.*;

import java.util.*;

import models.*;

public class Tareas extends Controller {

    public static void index(String filtro) {
       List<Tarea> tareas;
       if(filtro == null || filtro.isEmpty()){
              tareas = Tarea.findAll();
       }else{
              tareas = Tarea.findByTitulo( filtro );
       }
       renderArgs.put("filtro" , filtro);
       renderArgs.put("tareas" , tareas);
         render();
    }

    public static void addTarea(@Required @MinSize(value = 3) String nombre){
       if(validation.hasErrors()){
              flash.error("Por favor ingrese el nombre de la tarea.");
       }else{
              Tarea tarea = new Tarea(nombre).save();
       }
       index( null );
    }

    public static void verTarea(Long idTarea){
       Tarea tarea = Tarea.find( "byId" , idTarea).first();
       render( tarea );
    }

    public static void completarTarea(Long idTarea){
       System.out.println(">> "+idTarea);
       Tarea tarea = Tarea.find( "byId" , idTarea).first();
       tarea.completada = true;
       tarea.save();
       index( null );
    }

}
```

Este controlador MVC define las siguientes operaciones:

- ✔ **index()**, operación principal que obtiene todas las Tareas persistidas en base a un filtro, y las pone a disposición del motor de plantillas a través de los '**`renderArgs`**', finalmente recarga la página al llamar al método '**`render()`**' .
- ✔ **addTarea()**, operación que valida los parámetros recibidos (verificando que el valor '**`nombre`**' esté presente y tenga una longitud mínima de 3 caracteres) y persiste una nueva Tarea de no haber errores de validación, de haberlos retorna un mensaje de error. Finalmente recarga la página principal al llamar al método **`'index()`**'.

- **verTarea()**, operación que carga una Tarea dada y redirecciona el navegador hacia la ruta '**`tareas/vertarea`**' mostrando los valores de la tarea dada, y dando la opción de completarla si no se encuentra en ese estado.
- **completarTarea()**, permite completar una tarea dada cambiando su estado y persistiéndolo, posteriormente redirecciona al navegador hacia la página principal de Tareas.

Y configuramos una ruta para su contexto, para nuestro caso será **`'/tareas`**', en el archivo '**`conf/routes`**' agrega:

```
/tareas                 Tareas.index
```

En la carpeta '**`app/views'`** agrega un nuevo archivo con la ruta '**`app/views/Tareas/index.html`**' con el siguiente contenido:

```
#{extends 'main.html' /}
#{set title:'Tareas' /}

#{if flash.error}
    <p style="color:#c00">
        ${flash.error}
    </p>
#{/if}

<form action="@{Tareas.addTarea()}" method="GET">
      <input type="text" name="nombre"/>
      <input type="submit" value="Agregar tarea"/>
</form>

<hr/>

<h3> Filtro </h3>

<form action="@{Tareas.index()}" method="GET">
      <input type="text" name="filtro" value="${filtro}"/>
      <input type="submit" value="Filtrar tareas"/>
</form>


#{ifnot tareas}
      <p>No hay tareas</p>
#{/ifnot}

<ul>
#{list items:tareas , as:'tarea' }
      <li>
             ${tarea.titulo} - ${tarea.completada ? 'COMPLETADA' : 'PENDIENTE'}
             <a href="@{Tareas.verTarea( tarea.id )}"> Ver</a>
      </li>
#{/list}
</ul>
```

Esta plantilla define varios elementos, primeramente la cabecera extiende la página **`main.html`** heredando sus estilos y diseños. Posteriormente tenemos un contenido condicional que muestra los errores presentes en el componente **`Flash1`**.

A continuación tiene un formulario que ejecuta el coman**`do Tareas.addTarea()`** usando como parámetro el campo de texto nombre. Seguido a ésto se encuentran los controles de filtrado de tareas, que llama al mismo método que genera la página, solamente agrega un nuevo

parámetro filtro a la petición **`GET`**.

Finalmente tenemos la lista de Tareas que se generan condicionalmente, junto se coloca un enlace hacia la página ver Tarea.html que permite modificar Tareas de forma individual.

Y un archivo más con la ruta '**`app/views/Tareas/verTarea.html`**' con el siguiente contenido:

```
 #{extends 'main.html' /}
#{set title:'Tareas' /}

<h3> Tarea : ${tarea.titulo} - ${tarea.completada ? 'COMPLETADA' : 'PENDIENTE'} </h3>

#{ifnot tarea.completada}

<form action="@{Tareas.completarTarea()}" method="GET">
      <input type="text" name="idTarea" value="${tarea.id}" readonly="true"/>
      <input type="submit" value="Completar tarea"/>
</form>

#{/ifnot}

<a href="@{Tareas.index()}">Volver</a>

<hr/>
```

Esta vista extiende la página main.html usando todos sus componentes y estilos, adicionalmente tiene un contenido condicional que solamente es generado cuando la tarea recibida no está completada **`(#{ifnot})`**.

Dentro de este contenido condicional se encuentra un formulario que ejecuta la operación **`Tareas.completarTarea()`** pasando como parámetro el valor del campo de entrada **`idTarea`**.

Al final tenemos un enlace hacia la página principal generada, usando la llamada al método que renderiza dicha página.

Ahora inicia la aplicación con:

```
$> play run
```

y con el navegador web ingresa a '**`http://localhost:9000/tareas`**', se tendrá el siguiente resultado:

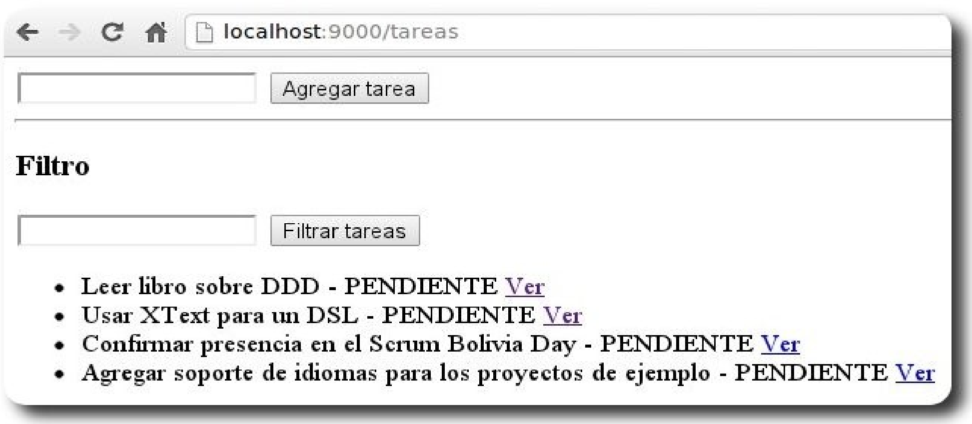


Figura 1. Pagina Principal

Proceder con las siguientes acciones:

- ✔ Agregue nuevas tareas usando los controles superiores, intente llamar a la acción sin colocar ningún valor y podrá verificar que las validaciones se activan.
- ✔ Al agregar varias tareas pruebe filtrar las tareas en usando los controles de filtrado, notará que la búsqueda es completa y no considera mayúsculas ni minúsculas.
- ✔ Complete algunas tareas a través del enlace '`Ver`':



Figura 3. Edición de tarea

Consideraciones importantes

- ✔ Todas las operaciones de los controladores son '`stateless`', es decir, no mantienen estados entre llamadas. La única forma de mantener estados es través de caché, parámetros y el componente Flash.
- ✔ El motor de plantillas usa Groovy para simplificar el código resultante, en versiones recientes Play! usa Scala como lenguaje para el motor de plantillas.
- ✔ Las rutas configuradas no requieren de los archivos HTML para funcionar, en caso de que los archivos relativos a cada controlador no estén presentes, tales operaciones se comportan como operaciones REST normales.
- ✔ Al igual que las plantillas los tags también son dinámicos. Play! permite que cada aplicación defina sus propios tags.
- ✔ Toda operación realizada de forma estática puede ser realizada usando Ajax, eso lo veremos en la tercera parte de esta serie ;-)

Quiere usar un IDE?

- ✔ **play ant**: Genera un archivo de construcción Ant para la aplicación.
- ✔ **play eclipsify**: Genera los archivos de configuración para Eclipse.
- ✔ **play netbeansify**: Genera los archivos de configuración para Netbeans.
- ✔ **play idealize**: Genera los archivos de configuración para IntelliJ.

Qué fué lo que hicimos?

Recopilemos lo que hicimos para nuestra aplicación Tareas:

1. Creamos la aplicación base.
2. Agregamos el elemento de dominio Tarea usando el componente base Model.
3. Habilitamos una base de datos en memoria a través de la configuración `db=mem`
4. Agregamos un componente controlador base denominado Tareas y

configuramos su ruta de mapeo HTTP en **`/conf/routes`**

5. Agregamos dos vistas de plantilla en '**`app/views/Tareas/index.html`**' y '**`app/views/Tareas/verTarea.html`**'
6. Verificamos la creación y modificación y borrado de tareas usando las vistas y controladores creados.

Conclusiones

Play! Provee poderosas capacidades con sus frameworks internos, en este apartado llegamos a ver los componentes y características básicas del motor de plantillas. Dichas plantillas son altamente configurables y extensibles de forma simple, sin requerimientos extravagantes ni agregados específicos.

Como punto clave se puede hacer notar la separación de contextos dentro de Play!, los controladores y rutas pueden trabajar de forma independiente a las vistas de plantilla, esto provee gran flexibilidad al momento de desarrollar aplicaciones REST.

Referencias

[1] http://www.playframework.org/documentation/1.2.5/home

[2] http://en.wikipedia.org/wiki/Web_2.0

Autor



Timoteo Ponce

Ingeniero de Software – Consultor Técnico

timo@gmail.com



Trabajando con Binary Large Object (BLOB) en PostgreSQL con GTK# y MonoDevelop II

Los BLOB (Binary Large Objects, objetos binarios grandes) son elementos utilizados en las bases de datos para almacenar datos de gran tamaño que cambian de forma dinámica.
Generalmente estos datos son imágenes, archivos de sonido y otros objetos multimedia; a veces se almacenan como BLOB código de binarios.



Introducción

Actualmente es importante que las aplicaciones dirigidas al usuario cuenten con una interfaz gráfica consistente que facilite el uso de la aplicación o del sistema de información, dentro de las opciones de Graphical User Interfaces (GUI) ó interfaces gráficas de usuario en el ecosistema .NET tenemos muchas alternativas: Windows Forms, WPF, Formularios ASP.NET, Silverlight, GTK# solo por mencionar algunas de las más utilizadas.

En el número 20 de la revista Atix **`http://revista.atixlibre.org/?p=259`** se publicó acerca de como leer y escribir datos binarios en PostgreSQL con C#, ahora en este documento complementamos la información anterior mostrando un programa en GTK# que hace uso de los mismos conceptos salvo que ahora los integra con las capacidades ofrecidas por Monodevelop para la creación de formularios GTK#. El ejemplo de este documento utiliza la misma tabla que se creó para el ejemplo del número anterior.

Diseñando la interfaz de usuario

Empezamos creando una nueva solución del tipo proyecto GTK# en Monodevelop:

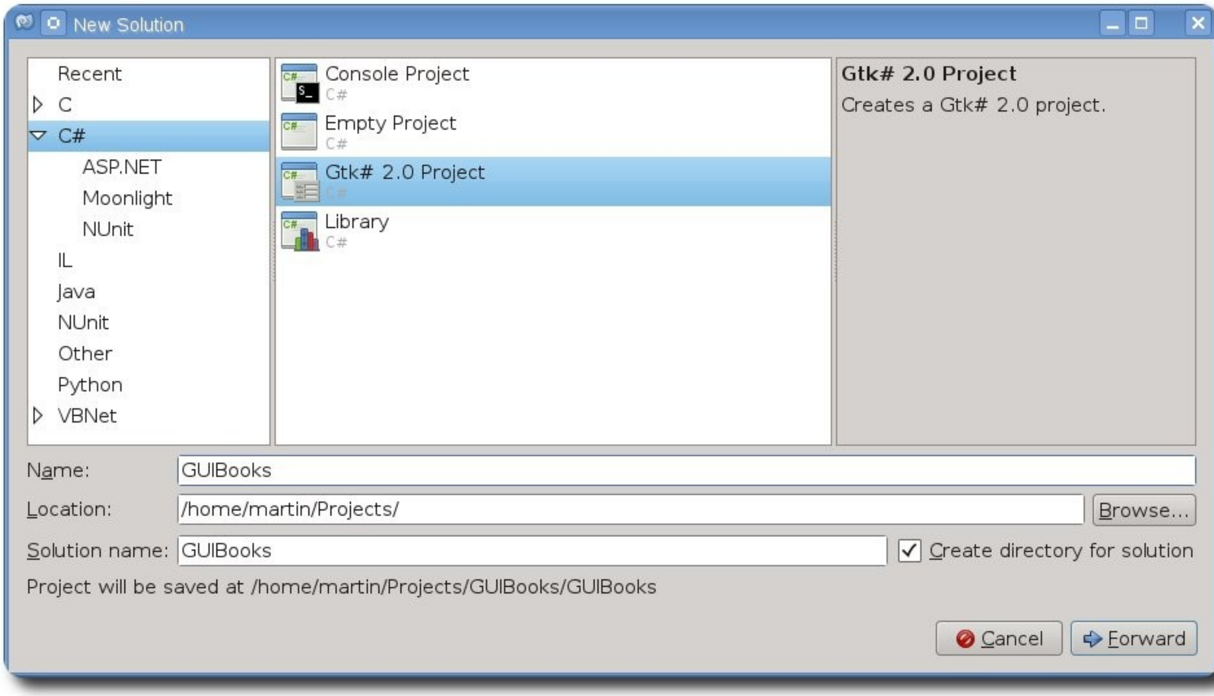


Figura 1. Monodevelop proyecto GTK#

Ahora diseñamos un formulario que incluya los siguientes elementos, como se muestra en la imagen:

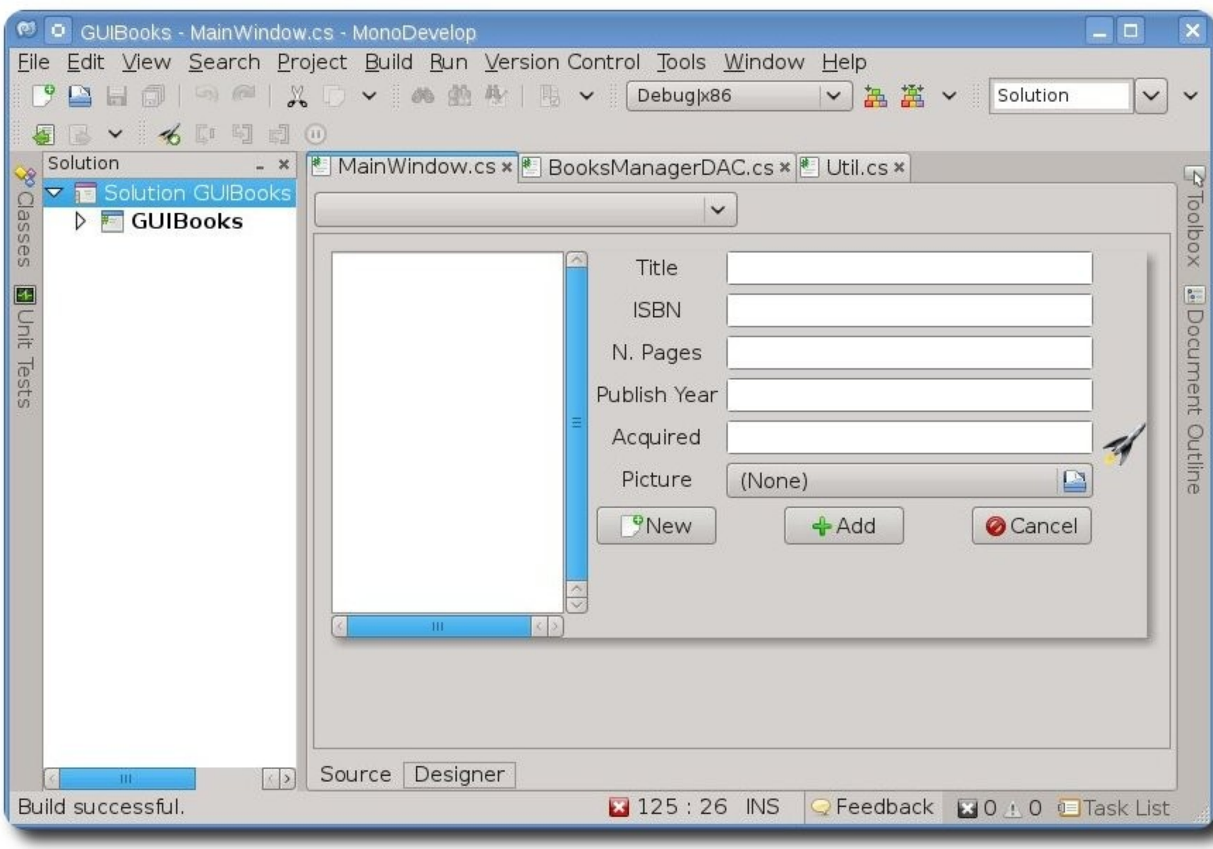


Figura 2. Diseño del formulario

La idea de está pantalla es poder tener el alta, la consulta y la edición de registros sin necesidad de tener una pantalla para cada operación, sino tener las operaciones CRUD (create, retrieve, update y delete) en un mismo formulario utilizando la visibilidad de los elementos. Es importante agregar el ensamblado **`Npgsql ()`** a la solución para el soporte a PostGreSQL, la solución deberá verse como en la siguiente imagen:

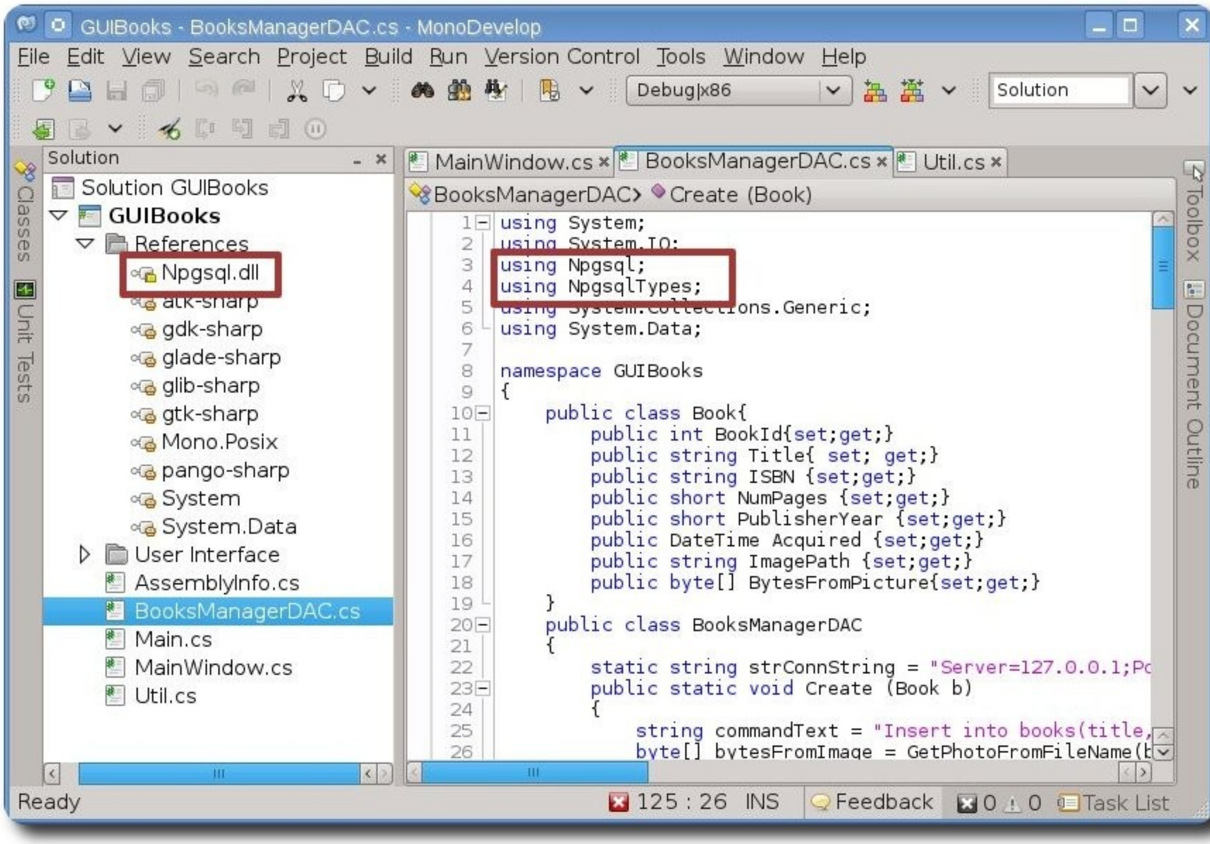


Figura 3. Muestra de la solución

El código de la clase BooksManagerDAC

```
using System;
using System.IO;
using Npgsql;
using NpgsqlTypes;
using System.Collections.Generic;
using System.Data;

namespace GUIBooks
{
public class Book{
public int BookId{set;get;}
public string Title{ set; get;}
public string ISBN {set;get;}
public short NumPages {set;get;}
public short PublisherYear {set;get;}
public DateTime Acquired {set;get;}
public string ImagePath {set;get;}
public byte[] BytesFromPicture{set;get;}
}
```

```
public class BooksManagerDAC
{
static string strConnString = "Server=127.0.0.1;Port=5432;Database=Books;User
ID=postgres;Password=Pa$$W0rd";
public static void Create (Book b)
{
      string commandText = "Insert into
books(title,isbn,numpages,publishyear,acquired,picture)Values(:title,:isbn,:numpages,:p
ublishyear,:acquired,:picture)";
      byte[] bytesFromImage = GetPhotoFromFileName(b.ImagePath);
      using(NpgsqlConnection conn = GetConnection())
      {
            using(NpgsqlCommand cmd = new NpgsqlCommand(commandText,conn))
            {
                  var paramTitle = new NpgsqlParameter("title",
NpgsqlDbType.Varchar);
            paramTitle.SourceColumn = "title";
            paramTitle.Value = b.Title;
            cmd.Parameters.Add(paramTitle);
            var paramISBN = new NpgsqlParameter("isbn", NpgsqlDbType.Varchar);
            paramISBN.SourceColumn = "isbn";
            paramISBN.Value = b.ISBN;
            cmd.Parameters.Add(paramISBN);
            var paramNumPages = new NpgsqlParameter("numpages", NpgsqlDbType.Smallint);
            paramNumPages.SourceColumn = "numpages";
            paramNumPages.Value = b.NumPages;
            cmd.Parameters.Add(paramNumPages);
            var paramPubYear = new NpgsqlParameter("publishyear",
NpgsqlDbType.Smallint);
            paramPubYear.SourceColumn = "publishyear";
            paramPubYear.Value = b.PublisherYear;
            cmd.Parameters.Add(paramPubYear);
            var paramAcquired = new NpgsqlParameter("acquired", NpgsqlDbType.Date);
            paramAcquired.SourceColumn = "acquired";
            paramAcquired.Value = b.Acquired;
            cmd.Parameters.Add(paramAcquired);
            var pPicture = new NpgsqlParameter("picture", NpgsqlDbType.Bytea);
            pPicture.SourceColumn = "picture";
            pPicture.Value = bytesFromImage;
            cmd.Parameters.Add(pPicture);
                  int r = cmd.ExecuteNonQuery();
                  Console.WriteLine("{0} affected",r);
                  }
      }
}

public static Book SelectById(int id){
      string commandText = "SELECT
bookid,title,isbn,numpages,publishyear,acquired,picture " +
            " FROM Books WHERE bookid = " + id.ToString();
      Book b = null;
      using(NpgsqlDataReader reader = GetReader(commandText))
      {
            int colBookId =     reader.GetOrdinal("bookid");
            int colTitle =      reader.GetOrdinal("title");
            int colISBN =reader.GetOrdinal("isbn");
            int colNumPages =   reader.GetOrdinal("numpages");
            int colPublishYear = reader.GetOrdinal("publishyear");
            int colAcquired = reader.GetOrdinal("acquired");
            int colPicture = reader.GetOrdinal("picture");
      while(reader.Read()){

                  b = new Book{
                        BookId = reader.GetInt32(colBookId),
                        Title = reader.GetString(colTitle),
```

```
                            ISBN = reader.GetString(colISBN)
                    };
                    if(!reader.IsDBNull(colNumPages))
                    b.NumPages = reader.GetInt16(colNumPages);
                    if(!reader.IsDBNull(colPublishYear))
                            b.PublisherYear = reader.GetInt16(colPublishYear);
                    if(!reader.IsDBNull(colAcquired))
                            b.Acquired = reader.GetDateTime(colAcquired);
                    if(!reader.IsDBNull(colPicture))
                            b.BytesFromPicture = GetPhotoForDataBase(reader,colPicture);
            }
        }
        return b;
}

static byte[] GetPhotoForDataBase(NpgsqlDataReader reader,int columnImage){
        byte[] imgBytes = (byte[])reader.GetValue(columnImage);
        MemoryStream memstream = new MemoryStream(imgBytes.Length);
        memstream.Write(imgBytes,0,imgBytes.Length);
        return memstream.ToArray();
}

static byte[] GetPhotoFromFileName(string filename){
        byte[] photo = null;
        using(FileStream fis = new FileStream(filename,FileMode.Open,FileAccess.Read))
        {
        BinaryReader reader = new BinaryReader(fis);
         photo = reader.ReadBytes((int)fis.Length);
                reader.Close();
        }
        return photo;
}

public static List<Book> SelectAll(){
        var resp = new List<Book>();
        Book b = null;
        string commandText = "SELECT bookid,title FROM Books ";
        using(NpgsqlDataReader reader = GetReader(commandText))
        {
                while(reader.Read()){
                        b = new Book();
                        b.BookId = Convert.ToInt32(reader["bookid"]);
                        b.Title = reader["title"].ToString();
                        resp.Add(b);
                }
        }
        return resp;
}
 static NpgsqlConnection GetConnection(){
        NpgsqlConnection conn = new NpgsqlConnection(strConnString);
        conn.Open();
        return conn;
}

static NpgsqlDataReader GetReader(string commandText){
        NpgsqlDataReader resp = null;
                NpgsqlConnection conn = GetConnection();
                        using (NpgsqlCommand cmd = new NpgsqlCommand(commandText, conn))
                        {
                                resp = cmd.ExecuteReader(CommandBehavior.CloseConnection |
                                            CommandBehavior.SequentialAccess);
                        }
                return resp;
}
}
}
```

La clase **Util** nos ayuda a encapsular la ventana emergente.

```
using System;
using Gtk;
namespace GUIBooks
{
	public class Util
	{

	public staticvoid ShowMessageBox(Window window,string msg){
	using (Dialog dialog = new MessageDialog (window,
	DialogFlags.Modal | DialogFlags.DestroyWithParent,
	MessageType.Info,
	ButtonsType.Ok,
	msg)) {
	dialog.Run ();
	dialog.Hide ();
	}
	}
	}
}
```

La clase **MainWindow** que es donde se encuentra toda la funcionalidad.

```
using System;
using Gtk;
using GUIBooks;
using System.Collections.Generic;

public partial class MainWindow: Gtk.Window
{
	ListStore store = null;
	List<Book> bookCollection = null;
	public MainWindow (): base (Gtk.WindowType.Toplevel)
	{
		Build ();
		gridView.Model = CreateModel();
	}

	protected void OnDeleteEvent (object sender, DeleteEventArgs a)
	{
		Application.Quit ();
		a.RetVal = true;
	}

	protected void OnBtnNewClicked (object sender, System.EventArgs e)
	{
		NewRecord(true);
	}

	protected void OnBtnAddClicked (object sender, System.EventArgs e)
	{
		Book b = new Book{
			Title = txtTitle.Text,
			ISBN = txtISBN.Text,
			NumPages = Convert.ToInt16(txtNumPages.Text),
			PublisherYear = Convert.ToInt16(txtPubYear.Text),
			Acquired = Convert.ToDateTime(txtAcquired.Text)
		};
```

```
            if(!string.IsNullOrEmpty(btnPicture.Filename))
                  b.ImagePath = btnPicture.Filename;
            try{
                  GUIBooks.BooksManagerDAC.Create(b);
                  Util.ShowMessageBox(this,"Record Inserted");
                  gridView.Model = CreateModel();
                  NewRecord(false);
            }catch(Exception ex){
                  Util.ShowMessageBox(this,ex.Message);
            }
      }

      ListStore CreateModel ()
      {
            btnAdd.Hide();
            btnPicture.Hide();
            CreateColumns();
            store = new ListStore(
                                          typeof(Int32),
                                          typeof(string)
            );
            bookCollection = BooksManagerDAC.SelectAll();
            foreach(Book item in bookCollection){
                  store.AppendValues(item.BookId,
                                    item.Title);
            }
            return store;
      }

      void CreateColumns(){
            CellRendererText rendererText = new CellRendererText ();
                  string[] colNames = {"Id","Title"};
                  for(int i = 0;i < colNames.Length;i++)
                        gridView.AppendColumn (new TreeViewColumn (
                                    colNames[i], rendererText, "text",i));
      }

      protected virtual void OnGridViewRowActivated (object o, Gtk.RowActivatedArgs
args)
      {
            Gtk.TreeIter iter;
            TreePath path = new TreePath(args.Path.ToString());

            store.GetIter(out iter,args.Path);
            int i = path.Indices[0];
            Book b = bookCollection.ToArray()[i] as Book;
            if(b != null)
                  ShowRecord(b.BookId);
      }

      void ShowRecord(int i){
            NewRecord(false);
            Book b = BooksManagerDAC.SelectById(i);
            txtTitle.Text = b.Title;
            txtISBN.Text = b.ISBN;
            txtNumPages.Text = b.NumPages.ToString();
            txtPubYear.Text = b.PublisherYear.ToString();
            txtAcquired.Text = b.Acquired.ToString();
            if(b.BytesFromPicture != null)
                  pictureBox.Pixbuf = new Gdk.Pixbuf(b.BytesFromPicture);
      }
```

```
      void NewRecord(bool isEditable){
            txtTitle.Text = txtISBN.Text =
            txtNumPages.Text = txtPubYear.Text =
            txtAcquired.Text = string.Empty;
            txtTitle.IsEditable = txtISBN.IsEditable =
            txtNumPages.IsEditable = txtPubYear.IsEditable =
            txtAcquired.IsEditable = isEditable;
            if(isEditable)
            {
                  btnAdd.Show();
                  btnNew.Hide();
                  btnPicture.Show();
                  pictureBox.Hide();
            }
            else
            {
                  btnPicture.Hide();
                  btnAdd.Hide();
                  btnNew.Show();
                  pictureBox.Show();
            }
      }
}
```

En sí todo el código es similar al que se público en el número anterior de la revista ATIX, lo que cambia es ya la integración con **GTK#** la cual se realiza en la clase **MainWindow** cuyo código es el siguiente, en donde explicaremos los puntos más importantes.

GTK# treeview y el patrón MVC

El componente GTK# Treeview sirve para representar datos de forma tabular o jerárquica, similar al control GridView en Windows Forms, este componente utiliza el patrón MVC (Model View Controller) que separa el modelo (datos) de la vista (la presentación), en este ejemplo los datos son representados por la lista genérica

```
List<Book> bookCollection = null;
```

En el método **CreateModel ()** asociamos la lista genérica con un objeto **ListStore** el cual implementa la interfaz **gtk.TreeModel** que representa un árbol jerárquico de columnas del mismo tipo (**strong-type**) por eso definimos el tipo de cada una de las columnas en el constructor del objeto **ListStore**.

```
ListStore CreateModel ()
      {
            btnAdd.Hide();
            btnPicture.Hide();
            CreateColumns();
            store = new ListStore(
                                          typeof(Int32),
                                          typeof(string)
            );
            bookCollection = BooksManagerDAC.SelectAll();
            foreach(Book item in bookCollection){
                  store.AppendValues(item.BookId,
                                    item.Title);
            }
            return store;
      }
```

Otro método importante es el **`CreateColumns()`** que se encarga de la creación de las columnas para el objeto **`TreeView`**, donde el render de los datos se lleva a cabo por el objeto **`CellRenderText`**

```
void CreateColumns(){
             CellRendererText rendererText = new CellRendererText ();
                    string[] colNames = {"Id","Title"};
                    for(int i = 0;i < colNames.Length;i++)
                           gridView.AppendColumn (new TreeViewColumn (
                                         colNames[i], rendererText, "text",i));
       }
```

Por último el método en donde se activa la selección de cada registro dentro del árbol

```
 protected virtual void OnGridViewRowActivated (object o, Gtk.RowActivatedArgs args)
       {
             Gtk.TreeIter iter;
             TreePath path = new TreePath(args.Path.ToString());

             store.GetIter(out iter,args.Path);
             int i = path.Indices[0];
             Book b = bookCollection.ToArray()[i] as Book;
             if(b != null)
                    ShowRecord(b.BookId);
       }
```

Este método tiene dos objetos básicos para acceder a un registro específico dentro del modelo de datos se trata de **`TreeIter`** y **`TreePath`**, **`TreeIter`** es la principal estructura para acceder a un **`TreeModel`** ya que se usa para referenciar un registro fijo dentro del modelo y para manipular el modelo agregando y quitando registros o modificando el contenido, como si fuese un cursor en base de datos[4] **`TreePath`** se refiere a una posición dentro del modelo y no a un registro fijo, por lo que se usa para traer el índice de un registro. TreePath representa posiciones como una cadena por ejemplo: "7:6:5" significa ve al séptimo nivel y ahí ve al sexto nodo hijo entonces quédate en el quinto nodo de ese nodo.

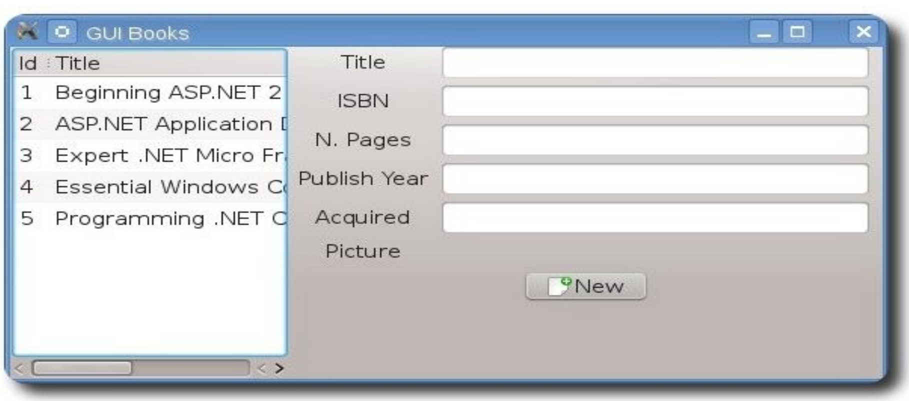


Figura 4. La aplicación ejecutándose

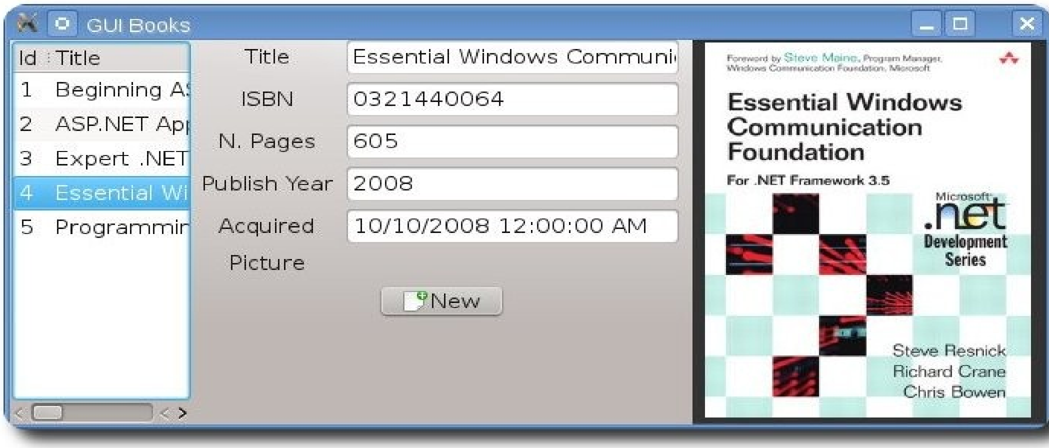


Figura 5. Consulta de un registro

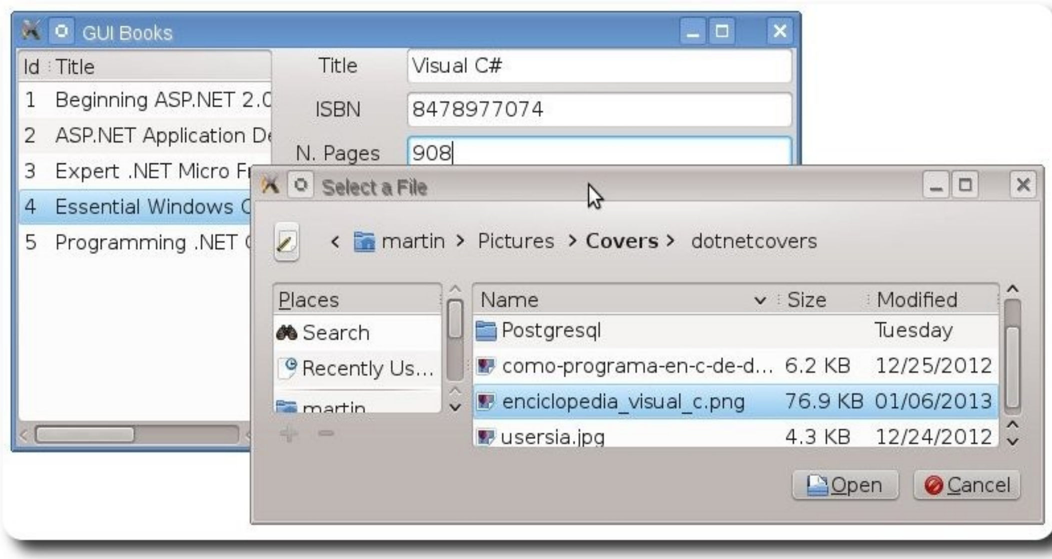


Figura 6. Alta de un registro

Conclusiones

Hoy día existen muchas alternativas para la construcción de interfaces gráficas, **GTK#** es una de las opciones a tomar en cuenta cuando se necesita decidir por un toolkit maduro y confiable dentro de mundo opensource, utilizando MonoDevelop nos ahorra tiempo para construir la interfaz gráfica por lo que podemos dedicar ese tiempo hacia toda la lógica del sistema, esto pensando en entregar aplicaciones que cumplan con los requerimientos del usuario, que es lo mismo a aplicaciones de buena calidad.

Referencias

[1] http://www.gtk.org/api/2.6/gtk/GtkTreeModel.html

[2] http://developer.gnome.org/gtk3/3.2/GtkTextIter.html

[3] Eric Harlow, Developing, Linux Applications with GTK+ and GDK, Prentice Hall 1999

[4] Niel M. BornStein, Edd Dumbill, Mono A Developer's Notebook, O'Really 2004

Autor



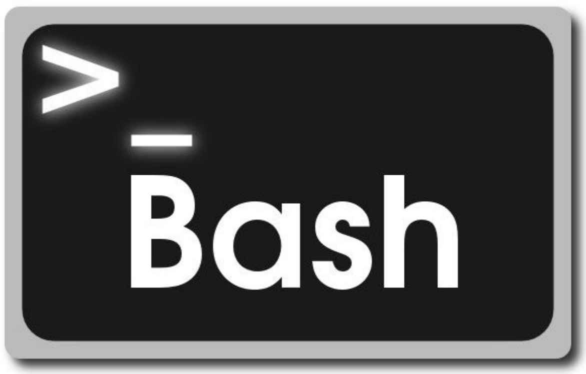
I.S. Martín Márquez

xomalli@gmail.com



Herramientas de un Desarrollador II: Shell

En el mundo de desarrollo de software, el uso de los frameworks se ha convertido en una herramienta poderosa, eficiente y eficaz al momento de afrontar proyectos de desarrollo de software, por su fácil aprendizaje, rápido desarrollo y estructura robusta.



Introducción

El Bourne Again Shell es el poderoso interprete de comandos por defecto en la mayoría de los sistemas UNIX y una herramienta indispensable de todo desarrollador.

Alguna vez la única manera de interactuar con un ordenador fue mediante comandos, hasta que se inventaron las interfaces gráficas. Hoy día abundan las GUIs y se han desarrollado programas para casi cualquier propósito, sin embargo, diferente a lo que muchos podrían pensar, la línea de comandos no ha muerto, grandiosas aplicaciones aún se desarrollan en esta modalidad, git, ffmpeg, vim, wicd, ncmpcpp, mplayer, alsamixer, frameworks como Ruby on Rails, Play Web y DJango, además del gran número de comandos clásicos para la administración del sistema operativo.

Como administradores y/o desarrolladores el intérprete de comandos no es opcional, es y seguirá siendo nuestro pan de cada día.

Bash es un intérprete de comandos y un lenguaje de programación, el Bourne Again Sheel es la versión del Bourne Shell creada por el proyecto GNU.

Este artículo no es un tutorial sobre comandos UNIX ni un manual de referencia con descripciones exhaustivas sino más bien una guía rápida específicamente a cerca de Bash. Al final proporciono algunas referencias para profundizar sobre los temas si a así lo desean.

Inicio de sesión

El primer paso a dar es invocar el comando bash, el cual obedece la sintaxis **`bash [opciones] [argumentos]`**, los elementos entre paréntesis son opcionales. Lo más probable es que Bash sea interprete de comandos por defecto y no tendrá que ejecutarlo ya que el emulador de terminal lo ejecuta por usted, aunque no está demás saber algunas cosas.

Opciones:

```
-i Crea un shell interactivo.
--norc
  No leer el archivo ~/.bashrc.
--posix
  Habilita el modo POSIX. Limita el
comportamiento de bash al estándar POSIX.
--rcfile archivo
  Indica a bash que lea la configuración
de inicio desde archivo y no desde
~/.bashrc
-, --
Termina el análisis de los parámetros.
```

Listado 1. Opciones del comando bash.

Si se indica el nombre de un archivo como argumento bash lo interpretará como un script(se verá más adelante) e intentará ejecutarlo.

Configuración de entorno

Bash admite personalización y lo hace mediante archivos de configuración:

- ✔ **`/etc/profile`**. Se ejecuta automáticamente al iniciar sesión.
- ✔ **`~/.bash_profile`**, **`~/.bash_login`** o **`~/.profile`**. El primer archivo que se encuentre, en este orden. Se ejecuta automáticamente al iniciar sesión.
- ✔ **`~/.bashrc`**. Se utiliza en en situaciones distintas al inicio de sesión.

A continuación muestro algunos ejemplos de configuración.

Personalizar el prompt

El prompt es el indicador que le precede a todo lo que tecleamos en la terminal, por ejemplo, **`rendon@intel:~$`** o **`root@intel:~#`**. El contenido del prompt se almacena en la variable de entorno PS1, veamos dos ejemplos:

El más común:

```
PS1='\u@\h:\w$ '
usuario@host:directorio_actual$
#resultado
```

Con colores:

```
PS1='[\e[0;32m][\t][\e[1;34m]\u[\e[0;37m]
[\e[1;37m]@\h~$ [\e[1;37m]'
```



Figura 1. Prompt con colores.

En la definición del prompt se han utilizados algunos valores especiales para obtener información del sistema, tales como la hora, nombre de usuario, host, directorio actual, entre otras.

```
\u Nombre del usuario actual.
\h Nombre del host.
\t La hora del sistema.
\d Fecha.
\w Directorio actual.
```

Listado 2. Valores especiales.

En [2] encontrarán bastante información sobre el prompt y los colores.

Variables de entorno

Las variables de entorno son datos que utiliza el sistema operativo o ciertas aplicaciones para ubicar recursos, por ejemplo, la variable **`PATH`** contiene un listado de directorios de donde el SO tiene que buscar programas, o **`JAVA_HOME`** que almacena la dirección del directorio de instalación de Java, PS1 que ya se vió en la sección anterior. El comando export permite declarar variables de entorno.

```
export JAVA_HOME="/usr/local/jdk1.7.0_09"
export PATH="$
{PATH}:/usr/local/misprogramas"
```

Alias

Los alias son a grandes rasgos atajos para comandos que nos ayudan a no teclear tanto o dar nombres más significativos a ciertas acciones, ejemplos:

```
alias apt-get="sudo apt-get"
alias grep="grep --color=always"
alias ls='ls $LS_OPTIONS'
```

Al intentar ejecutar grep lo que realmente se ejecuta es **`grep --color=always`** y así con los demás comandos.

Modos de edición

Bash tiene dos modos de edición, emacs y vi, que permiten trabajar de manera más eficiente y cómoda con el intérprete. El modo por defecto es emacs. Una de las ventajas de estos modos de edición es que permiten trabajar sin retirar las manos de su posición básica.

Para habilitar estos modos de edición emplea los siguientes comandos, ya sea en modo interactivo o en el archivo de configuración

para hacer el cambio permanente.

```
set -o vi
set -o emacs
```

En [3] y [4] puedes encontrar una lista detallada de comandos para los modos vi y emacs respectivamente.

Redireccionamiento

El funcionamiento básico de todo programa es recibir una entrada, procesar los datos y producir una salida. En los sistemas UNIX existen dos tipos de salida, normal y de errores. Por defecto la entrada corresponde al teclado y la salida a la pantalla.

Descriptor de archivo	Nombre	Abreviación	Por defecto
0	Entrada estándar	stdin	Teclado
1	Salida estándar	stdout	Pantalla
2	Error estándar	stderr	Pantalla

Tabla 1. Entradas y salidas en UNIX.

La redirección de datos permite modificar el comportamiento normal de los programas con respecto a la entrada y salida de datos, a continuación un resumen con ejemplos de algunos tipos de redirección:

comando > archivo

Redirecciona la salida de comando a archivo(el archivo se sobrescribe).

Ej.

```
$ ls -1 > salida.txt
```

comando < archivo

El comando toma la entrada desde archivo.

Ej.

```
$ bc < entrada.txt
```

comando >> archivo

Redirecciona la salida de comando a archivo (el archivo no se sobrescribe, se agrega al final).

Ej.

```
$ ls imagenes/ > archivos.txt
$ ls videos/ >> archivos.txt
```

archivos.txt contiene el listado de archivos en imágenes y también los del directorio videos.

comando <<marca

Conocido como here document, todo texto que se tecleé después de marca hasta encontrar nuevamente marca se tomará como la entrada del programa, marca no forma parte de la entrada.

Ej.

```
$cat > salida.txt <<EOF
> linea 1
> linea 2
>EOF
```

El texto entre **EOF** y **EOF** se almacena en el archivo **salida.txt**.

comando <<< cadena

Conocido como here string, **cadena** es tomada como entrada de comando, muy útil.

Ej.

```
$ bc <<< "(2^10 + 6)/2"
```

comando n>archivo

Redirecciona la salida que normalmente iría al descriptor de archivo n a archivo.

Ej.

```
$ gcc > error.txt
  gcc: fatal error: no input files
  compilation terminated.
```

```
$ cat error.txt
$
$ gcc 2> error.txt
$ cat error.txt
  gcc: fatal error: no input files
  compilation terminated.
```

El primer ejemplo de redirección no funciona y el archivo **`error.txt`** esta vació porque los mensajes de error de gcc van dirigidos al descriptor de errores(2) y no al descriptor de salida(1). El segundo ejemplo logra su objetivo.

comando <&n

Toma la entrada para comando desde el descriptor de archivo n.

Ej.

```
$ echo "(3 * 4)/2" > entrada.txt
$ exec 5<entrada.txt
$ bc <&5
6
```

comando &> archivo

Envía la salida estándar y errores a archivo.

comando > salida.txt 2> errores.txt

Envía la salida a salida.txt y los errores a **`errores.txt`**.

comando | tee salida.txt

Envía la salida de comando a tee para que lo muestre en pantalla y a la ves lo almacene en el archivo **`salida.txt`**.

Ten cuidado con los espacios, no todas las redirecciones permiten espacios antes o después de < y >.

Tuberías y substitución de procesos

comando1 | comando2

comando2 toma como entrada la salida generada por comando1.

Ej.

```
$ ls -1 | grep ".*txt"
$ echo "1 9 7 1 2 9 2 1" | tr ' ' '\n'
  |  sort -n | uniq
```

comando <(comandos)

comando toma como entrada la salida de comandos.

Ej.

```
$ uniq <(echo "1 9 7 1 2 9 2 1" | tr ' ' 
'\n'
  |  sort -n)
```

Bash como lenguaje de programación

En este apartado vamos a ver algunas de las características de Bash que le permiten funcionar como un lenguaje de programación. Cabe decir que Bash no es un lenguaje de propósito general sino más bien orientado a la administración de sistemas y automatización de tareas.

Variables

Una variable es un identificador que almacena cierto valor, veamos como definir una.

```
$ nombre="Rafael"
$ declare -i edad=20+3 #variables de tipo 
entero
$ idiomas=(español ingles latín hebreo 
chino)
```

Las reglas para nombrar variables son las mismas que en la mayoría de los lenguajes: letras, guion bajo, números, el nombre no debe iniciar con número.

Para recuperar el valor de una variable anteponga un $ al nombre de la variable.

```
$ echo "Mi nombre es $nombre y tengo 
$edad años y hablo ${idiomas[@]}"
```

Variables predefinidas

Bash define algunas variables por defecto, muy útiles cuando trabajamos con scripts.

```
$# Número de argumentos pasados a un
comando.
$? Valor de salida del último comando.
$$ Número de proceso del shell.
$0 Nombre del comando que se esta
ejecutando.
$n Valores de los argumentos. $1 es el
primer parámetro, $2 el segundo
parámetro, etc.
"$*"  Todos los argumentos como una
cadena ("$1 $2 ...").
"$@" Todos los argumentos entre comillas
dobles de forma individual.
```

Arreglos

Bash provee dos tipos de arreglos, basados en índices (los índices son números) y asociativos (los índices son cadenas).

```
$ idiomas=(español ingles latín hebreo
chino)
$ echo ${idiomas[0]}
español
$ echo ${idiomas[1]}
ingles
$ declare -A P=( [mexico]=52
[bolivia]=591 [canada]=11 )
$ echo ${P[mexico]}
52
```

Operaciones sobre arreglos:

```
${arreglo[i]} Valor en el índice i
${arreglo[*]} Todos los elementos
${arreglo[@]} Todos los elementos
${#name[*]}   Número de elementos en el
arreglo
${#name[@]}   Número de elementos en el
arreglo
```

Expresiones aritméticas

Las operaciones aritméticas se puede realizar con **`$(( expresión ))`**.

```
$ echo $(( (2**10)/2 ))
512
$ a=3; b=10;
$ echo $(( a += b ))
$ echo $a
13
```

Los operadores en su mayoría son los mismos que en C/C++, salvo algunas excepciones, la exponenciación(**), por ejemplo.

Estructuras de control

Las estructuras de control permiten guiar el flujo de ejecución de un programa. Bash provee if, if ... else y if ... elif ... else.

If

```
if COMANDOS-DE-PRUEBA
then
  BLOQUE-DE-COMANDOS
fi
```

Ej.

```
if [ $edad -lt 18 ]
then
  echo "Fuera de aquí. Solo para
adultos.";
fi
```

If … else

```
if COMANDOS-DE-PRUEBA
then
  BLOQUE-DE-COMANDOS-1
else
  BLOQUE-DE-COMANDOS-2
fi
```

Ej.

```
if [ $edad -ge 18 ]
then
  echo "Adelante.";
else
  echo "Fuera de aquí. Solo para
adultos.";
fi
```

If … elif … else

```
if COMANDOS-DE-PRUEBA
then
  BLOQUE-DE-COMANDOS-1
elif COMANDOS-DE-PRUEBA
  BLOQUE-DE-COMANDOS-2
fi
```

Ej.

```
if [ $edad -ge 18 ]
then
  echo "Adelante.";
elif [ $influyente ]
  echo "Adelante por favor";
```

```
else
  echo "Fuera de aquí. Solo para
adultos.";
fi
```

Para las comparaciones he utilizado algunos operadores diferentes a los convencionales, como son

- lt Menor que.
- ge Mayor o igual a.

Véase man para más información sobre los operadores de comparación.

Estructuras de repetición

Bash soporta las estructuras while y for para repetición de operaciones. Su sintaxis es la siguiente.

For

```
for argumento in [lista]
do
 comando(s)...
done
```

Ej.

```
for i in {1..10}
do
  echo $(( i**2 ));
done
```

While

```
while [ condición ]
do
 comando(s)...
done
```

Ej.

```
while read archivo
do
  rm "$archivo";
done < <(ls -1)
```

Scripts

Un script no es más que una serie de comandos y operaciones guardadas en un archivo, listos para ser ejecutados por Bash. La estructura de un script es la siguiente.

```
#!/bin/bash
comando-1;
comando-2;
.
.
comando-n;
exit 0;
```

La primer línea especifíca con que comando deben ser ejecutados los comandos en el archivo. El contenido de la primera línea es conocido como shebang.

Para finalizar les dejo un pequeño script donde se emplean algunos elementos ya vistos, es un script muy sencillo que empleo para configurar un directorio con plantillas de código en diferentes lenguajes para usarse en concursos de programación.

```
#!/bin/bash
PC="/home/data/cs/pc";

if    [ $1 = "codeforces" ]; then list=({A..E});
elif  [ $1 = "cookoff"    ]; then list=({A..E});
elif  [ $1 = "challenge"  ]; then list=({A..J});
elif  [ $1 = "hackercup"  ]; then list=({A..C});
fi;

for problem in ${list[@]}
do
  mkdir $problem;
  cd $problem;
  cp "$PC/TEMPLATE/template.cpp" "$problem.cpp";
  cp "$PC/TEMPLATE/template.rb"  "$problem.rb";
  cp "$PC/TEMPLATE/template.py"  "$problem.py";
```

```
  sed "s/Main/$problem/g"\
        < "$PC/TEMPLATE/Main.java"\
        > "$problem.java";
  cd ..;
done

if [ $1 = "-f" ]
then
  if [ -z $2 ]
  then
    echo "Provide a file name!";
    exit;
  fi

  # file name
  fn=$(sed 's/\([^.]\+\).*$/\1/g' <<< $2);
  # file type
  ft=$(sed 's/[^.]\+.\(.\+\)$/\1/g' <<< $2);

  if [ $ft != "cpp" -a $ft != "java"\
      -a $ft != "py" -a $ft != "rb" ]
  then
    echo "File type($ft) not supported :(!";
    exit;
  fi

  if [ $ft = "java" ]
  then
    sed "s/Main/$fn/g"\
        < "$PC/TEMPLATE/Main.java" > "$fn.$ft";
  else
    cp "$PC/TEMPLATE/template.$ft" "$fn.$ft";
  fi
fi
```

Script 1: setcontest.sh

Existen básicamente dos formas de ejecutar un script. Invocar bash pasando como argumento el el archivo o dando permisos de ejecución al script, y se ejecuta como cualquier otro comando.

```
$ bash mi_script.sh      # primera forma

$ chmod +x mi_script.sh  # segunda forma
$ ./mi_script.sh
```

Conclusión

Bash es una herramienta poderosa y extensa, hablar a profundidad del tema involucraría cientos de páginas y ciertamente el autor de este artículo no cuenta con el conocimiento suficiente. En este documento solo hemos visto algunas de las características básicas pero que considero suficiente para iniciarse. Además recalcar que lo antes visto es unicamente referente a Bash, en futuras publicaciones escribiré sobre ciertos programas que agregan funcionalidad a la línea de comandos.

Referencias

[1] Bash Pocket reference, Arnold Robbins, O'Reily.

[2] Color Bash Prompt, https://wiki.archlinux.org/ index.php/Color_Bash_Prompt

[3] Working Productively in Bash's, Vi Command Line Editing Mode,
http://www.catonmat.net/blog/bash-vi-editing-mode-cheat-sheet/

[4] Bash Emacs Editing Mode Cheat Sheet
http://www.catonmat.net/blog/bash-emacs-editing-mode-cheat-sheet/

Autor



Rafael Rendón Pablo

Estudiante de Ingeniería en Sistemas Computacionales

smart.rendon@gmail.com



Warnings
Alertas no fatales

Entregar alertas no fatales al usuario, acerca de problemas encontrados cuando se ejecuta un programa

El módulo warnings fue introducido en PEP 230 como una manera de advertir a los programadores sobre cambios en el lenguaje o de las características de la biblioteca en anticipación a cambios incompatibles que vienen con Python 3.0. Dado que las advertencias no son fatales, un programa puede encontrar las mismas situaciones que producen advertencias muchas veces en el curso de la ejecución. El módulo warnings suprime las advertencias repetidas del mismo código fuente para reducir la molestia de ver el mismo mensaje una y otra vez. Puedes controlar los mensajes impresos de caso por caso utilizando la opción `-W` para intérprete o llamando funciones que se encuentran en warnings desde tu código.

Categorías y filtrado

Las advertencias son categorizadas usando sub clases de la clase de excepción incorporada Warning. Una serie de valores estándar están descritos en la documentación del módulo, y puedes añadir tus propios heredando de Warning para crear nuevas clases.

Los mensajes son filtrados usando ajustes controlados por la opción `-W` para el intérprete. Un filtro consiste de cinco partes, la acción, el mensaje, la categorías, el módulo y el número de línea. Cuando una advertencia es generada, ésta es comparada con los filtros que han sido registrados. El primer filtro que coincide controla la acción tomada para la advertencia. Si ningún filtro coincide, la acción por defecto es tomada.

Las acciones que entiende el mecanismo de filtro son:

- ✔ **error**: Convierte la advertencia en un excepción.
- ✔ **ignore**: Descarta la advertencia.
- ✔ **always**: Siempre emite una advertencia.
- ✔ **default**: Imprime la advertencia la primera vez que es generada desde cada lugar.
- ✔ **module**: Imprime la advertencia la primera vez que es generado desde cada módulo.
- ✔ **once**: Imprime la advertencia la primera vez que es generada.

El mensaje del filtro es una expresión regular que es usada para comparar el texto de la advertencia.

La categoría del filtro es el nombre de la clase de excepción, descrita con anterioridad.

El módulo contiene una expresión regular que se compara con el nombre del módulo que genera la advertencia.

El número de línea puede ser usado para cambiar el manejo de ocurrencias específicas de una advertencia. Usa 0 para aplicar el filtro a todas las ocurrencias.

Generando advertencias

La manera más simple de emitir una advertencia desde tu código es ejecutar **warn()** con el mensaje como argumento:

```
import warnings

print 'Antes de la advertencia'
warnings.warn('Este es un mensaje de advertencia')
print 'Después de la advertencia'
```

Entonces cuando tu programa ejecuta, el mensaje es impreso:

```
$ python warnings_warn.py

warnings_warn.py:13: UserWarning: Este es un mensaje de advertecia
  warnings.warn('Este es un mensaje de advertencia')
Antes de la advertencia
Después de la advertencia
```

A pesar de que la advertencia es impresa, el comportamiento por defecto es continuar después de la advertencia y ejecutar el resto del programa. Podemos cambiar este comportamiento con un filtro:

```
import warnings

warnings.simplefilter('error', UserWarning)

print 'Antes de la advertencia'
warnings.warn('Este es un mensaje de advertencia')
print 'Después de la advertencia'
```

Este filtro instruye al módulo warnings elevar una excepción cuando la advertencia es emitida.

```
$ python warnings_warn_raise.py

Before the warning
Traceback (most recent call last):
  File "warnings_warn_raise.py", line 15, in <module>
    warnings.warn('Este es un mensaje de advertencia')
UserWarning: Este es un mensaje de advertencia
```

También podemos controlar el comportamiento desde la línea de comando. Por ejemplo, si volvemos a **warnings_warn.py** y cambiamos el filtro para elevar un error en **UserWarning**, vemos la excepción.

```
$ python -W "error::UserWarning::0" warnings_warn.py

Before the warning
Traceback (most recent call last):
  File "warnings_warn.py", line 13, in <module>
    warnings.warn('Este es un mensaje de advertencia')
UserWarning: Este es un mensaje de advertencia
```

Desarrollo

Ya que dejé los campos para mensaje y módulo vacíos, fueron interpretadas coincidiendo con cualquier cosa.

Filtrando con patrones

Para filtrar en reglas más complejas, usa **`filterwarnings()`**. Por ejemplo, para filtrar en base al contenido del texto del mensaje:

```
import warnings

warnings.filterwarnings('ignore', '.*no muestres.*',)

warnings.warn('Muestra este mensaje')
warnings.warn('No muestres este mensaje')
```

El patrón contiene "**`no muestres`**", pero el mensaje real usa "**`No muestres`**". El patrón coincide porque la expresión regular siempre es compilada para buscar coincidencias con mayúsculas y minúsculas.

```
$ python warnings_filterwarnings_message.py

warnings_filterwarnings_message.py:14: UserWarning: Muestra este mensaje
  warnings.warn('Muestra este mensaje')
```

Ejecutando este código desde la línea de comando:

```
import warnings

warnings.warn('Muestra este mensaje')
warnings.warn('No muestres este mensaje')
```

Resulta:

```
$ python -W "ignore:do not:UserWarning::0" warnings_filtering.py

warnings_filterwarnings_message.py:14: UserWarning: Muestra este mensaje
  warnings.warn('Muestra este mensaje')
```

Las mismas reglas del patrón se aplican al nombre del módulo que contiene la llamada de advertencia. Para suprimir todas las advertencias desde el módulo **`warnings_filtering`**:

```
import warnings

warnings.filterwarnings('ignore',
                        '.*',
                        UserWarning,
                        'warnings_filtering',
                        )

import warnings_filtering
```

Ya que el filtro está en su lugar, no se emiten advertencias cuando **`warnings_filtering`** es importado:

```
$ python warnings_filterwarnings_module.py
```

Para suprimir solo la advertencia en la línea 14 de **warnings_filtering**:

```
import warnings

warnings.filterwarnings('ignore',
                        '.*',
                        UserWarning,
                        'warnings_filtering',
                        14)

import warnings_filtering
```

```
$ python warnings_filterwarnings_lineno.py
```

Advertencias repetidas

Por defecto, la mayoría de las advertencias solo se imprimen la primera vez que ocurren en una ubicación, siendo ubicación la combinación de módulo y número de línea.

```
import warnings

def funcion_con_advertencias():
    warnings.warn('Esta es una advertencia!')

funcion_con_advertencias()
funcion_con_advertencias()
funcion_con_advertencias()
```

```
$ python warnings_repeated.py

warnings_repeated.py:13: UserWarning: Esta es una advertencia!
  warnings.warn('Esta es una advertencia!')
```

La acción once puede ser usada para suprimir instancias del mismo mensaje desde diferentes ubicaciones.

```
import warnings
warnings.simplefilter('once', UserWarning)

warnings.warn('Esta es una advertencia!')
warnings.warn('Esta es una advertencia!')
warnings.warn('Esta es una advertencia!')
```

```
$ python warnings_once.py
warnings_once.py:14: UserWarning: Esta es una advertencia!
  warnings.warn('Esta es una advertencia!')
```

De manera similar, module suprimirá mensajes repetidos desde el mismo módulo, sin importar el número de línea.

Funciones alternativas de entrega de mensajes

Normalmente advertencias se imprimen a sys.stderr. Puedes cambiar ese comportamiento reemplazando la función **showwarning()** dentro del módulo warnings. Por ejemplo, si quieres que las advertencias vayan a un archivo de registro en vez de stderr, puedes reemplazar **showwarning()** con una función como ésta:

```
import warnings
import logging

logging.basicConfig(level=logging.INFO)

def manda_advertencias_al_registro(message, category, filename, lineno, file=None):
    logging.warning(
        '%s:%s: %s:%s' %
        (filename, lineno, category.__name__, message))
    return

old_showwarning = warnings.showwarning
warnings.showwarning =  manda_advertencias_al_registro

warnings.warn('Este es un mensaje de advertencia')
```

Así, cuando **warn()** es ejecutada, las advertencias son emitidas con el resto de los mensajes de registro.

```
$ python warnings_showwarning.py

WARNING:root:warnings_showwarning.py:24: UserWarning: Este es un mensaje de advertencia
```

Formato

Si está bien que las advertencias vayan a **stderr**, pero no te gusta el formato, puedes reemplazar **formaatwarning()**.

```
import warnings

def advertencia_en_una_linea(message, category, filename, lineno, file=None,
line=None):
    return ' %s:%s: %s:%s' % (filename, lineno, category.__name__, message)

warnings.warn('Este es un mensaje de advertencia, antes')
warnings.formatwarning = advertencia_en_una_linea
warnings.warn('Este es un mensaje de advertencia, después')
```

```
$ python warnings_formatwarning.py

warnings_formatwarning.py:15: UserWarning: Este es un mensaje de advertencia, antes
  warnings.warn('Este es un mensaje de advertencia, después')
 warnings_formatwarning.py:17: UserWarning: Este es un mensaje de advertencia, después
```

Niveles de pila en advertencias

Notarás que por defecto el mensaje de advertencia incluye la línea de código que lo generó, cuando está disponible. Pero no es del todo útil ver la línea de código con mensaje de advertencia real. En su lugar, puedes decir a **warn()** qué tan lejos tiene que ir para encontrar la línea que ejecutó la función que contiene la advertencia. De ésta manera los usuarios de una función obsoleta ven dónde se ejecuta la función en vez de la implementación de la función.

```
import warnings

def funcion_vieja():
    warnings.warn(
        'funcion_vieja es vieja, usa funcion_nueva en su lugar',
        stacklevel=2)
```

```
def ejecuta_funcion_vieja():
    funcion_vieja()

ejecuta_funcion_vieja()
```

Observa que en este ejemplo **`warn()`** necesita subir la pila dos niveles, uno para si misma y otra para **`funcion_vieja()`**.

```
$ python warnings_warn_stacklevel.py

warnings_warn_stacklevel.py:18: UserWarning:  funcion_vieja es vieja, usa funcion_nueva
en su lugar
  funcion_vieja()
```

Autor



Ernesto Rico Schmidt

Usuario de Linux y Software Libre desde 1994
e.rico.schmidt@gmail.com



Pandora FMS 4.0.3

Una nueva versión de Pandora FMS llega con el comienzo del nuevo año. Pandora FMS 4.0.3 es una actualización menor con nuevas características y la mayoría de los bugs reparados para hacer la versión más fiable.

Esta versión tiene dos nuevas características : instantáneas de comandos y comparación de gráficas.

La funcionalidad de instantáneas de comandos (command snapshot) permite generar una imagen estática de cualquier comando de salida del sistema. Si algo va mal, command snapshot accede a información detallada sobre el problema y la ofrece sin que se tenga que ejecutar ningún comando para averiguar qué está pasando. Por ejemplo, se podría comparar el valor del uso del CPU con una instantánea del comando “top”. Está nueva funcionalidad también almacena instantáneas en la base de datos, mismas que se pueden consultar en cualquier momento.

La nueva funcionalidad de comparación de gráficas (graphic comparison) genera dos gráficas con las que se puede comparar intérvalos de tiempo. Esta característica ofrece dos opciones para presentar las gráficas: solapadas o separadas. Esta función será muy útil para comparar datos y tendencias mediante la comparación de un rango determinado de tiempo con el intérvalo previo, cuando se haya realizado un gran cambio en el sistema. Se podría comparar esta semana con la semana pasada o con el mes pasado.

Nuestro objetivo principal es ofrecer el mejor sistema de monitorización del mercado. Estamos comprometidos para mejorar la fiabilidad y el rendimiento de Pandora FMS. Por este motivo, esta nueva versión 4.0.3 llega con muchos de los bugs reparados y con las funcionalidades mejoradas que le mencionamos continuación.

Consola

- ✔ Mapas GIS mejorados. Ahora muestran un icono del agente y el nombre del agente se puede ocultar.

- Vistas de eventos más rápidas y varios bugs reparados en relación a la validación de eventos.
- Gestión de la base de datos mejorada para permitir que la base de datos principal y la base de datos histórica puedan alojarse en el mismo host.
- Integración de la gestión de incidentes entre Pandora FMS e Integria IMS mejorada.
- Varios problemas relacionados con el sistema ACL han sido reparados.
- Visualización de la red de mapas
- Varios problemas sobre la gestión de las políticas y la visualización han sido reparados.
- Soporte añadido para mover agentes a través de las metaconsolas.

Servidor

- Soporte mejorado para SNMP.
- Creación de nuevos plugins y mejora de los existentes.
- Mejor rendimiento de los servidores de la red enterprise tras la reparación de algunos problemas.
- Detección de la Library SSL mejorada en la monitorización transaccional de la web enterprise.

Agente

Un problema en el sistema Solaris y en la mecánica de las comunicaciones ha sido solucionado.

El módulo del **`log-event`** para Windows se ha actualizado para acoplarse a la nueva estructura de eventos.

Soporte para Windows 8

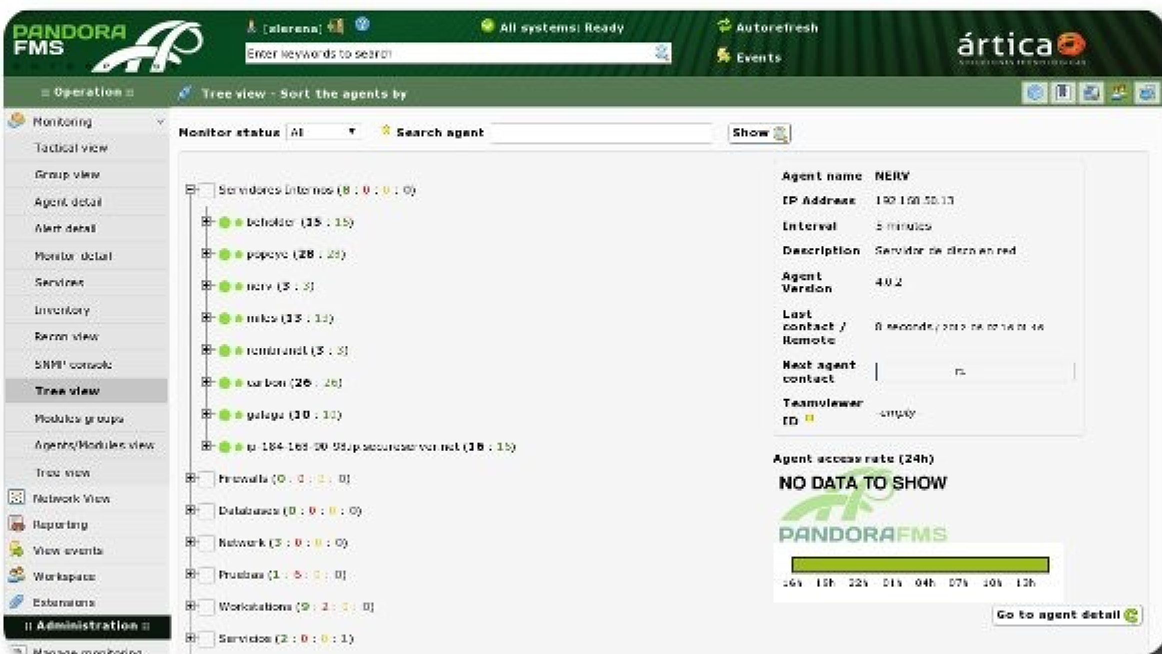
Para descargar Pandora FMS, puedes hacer clic en el siguiente enlace **`http://pandorafms.com/Community/download/`**. y podrás descargar pandora FMS con diferentes soportes (ISO, VMware image, DEB, RPM y TAR.GZ). Si deseas obtener más información sobre Pandora FMS, échale un vistazo a la documentación **`http://pandorafms.com/Community/doc/`** o a la página web : **`http://pandorafms.com/Home/Home/es`**

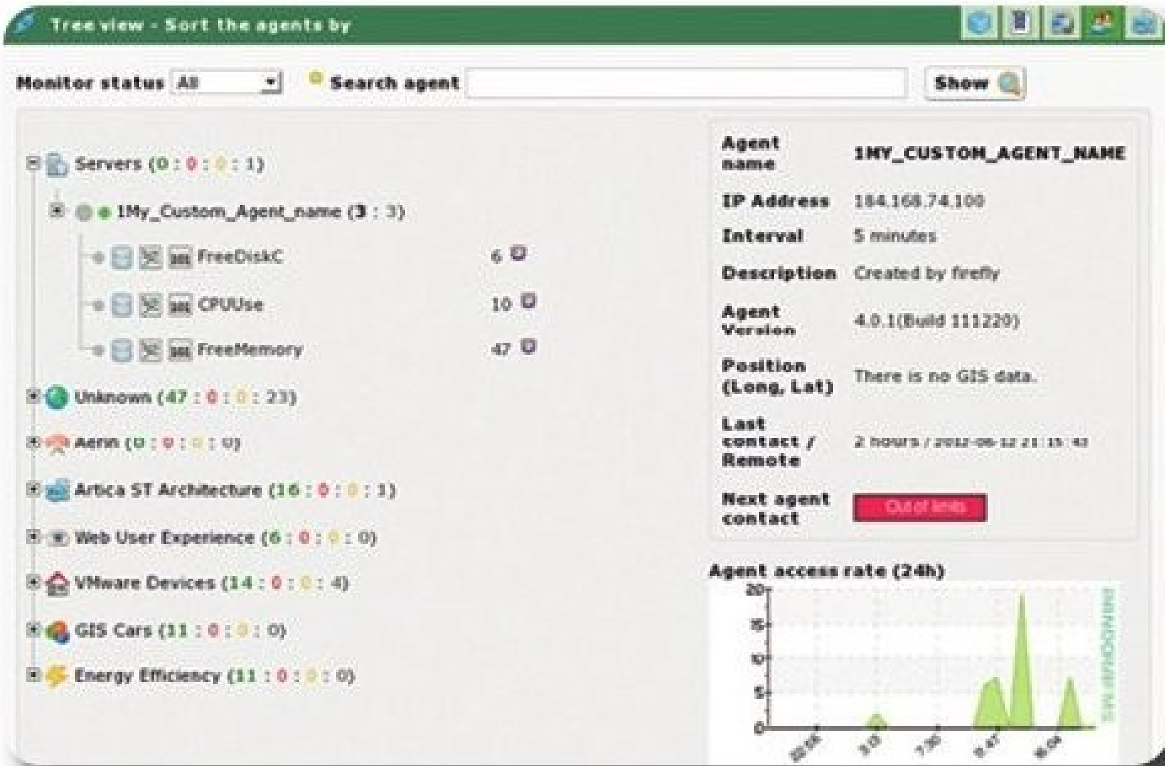
Capturas de Pantalla



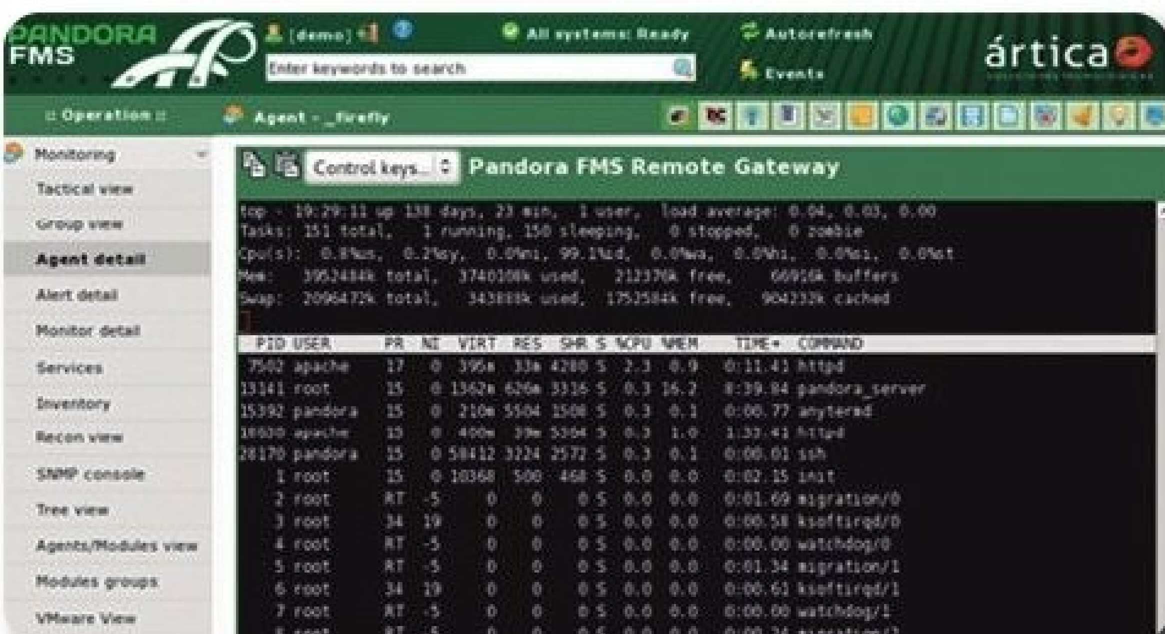



PANDORA FMS
[admin]
All systems: Ready
Autorefresh
Events
Enter keywords to search
:: Operation ::
Tactical view
Updated at realtime
Monitoring
Tactical view
Group view
Agent detail
Alert detail
Monitor detail
Services
Inventory
Recon view
SNMP console
Tree view
Modules groups
Agents/Modules view
Tree view
Network View
Reporting
View events
Workspace
Extensions
Global health
Monitor health
Module sanity
Alert level
Monitor checks
5672
Monitors critical
105
Monitors warning
1
Monitors normal
5437
Monitors unknown
129
Monitors not init
Alerts defined
Latest events
V.
S.
Type
Event name
Agent name
Timestamp
Tactical server information
Name
Type
Status
Load
Lag
(Data)
(Network)
(Snmp)
100%
N/A

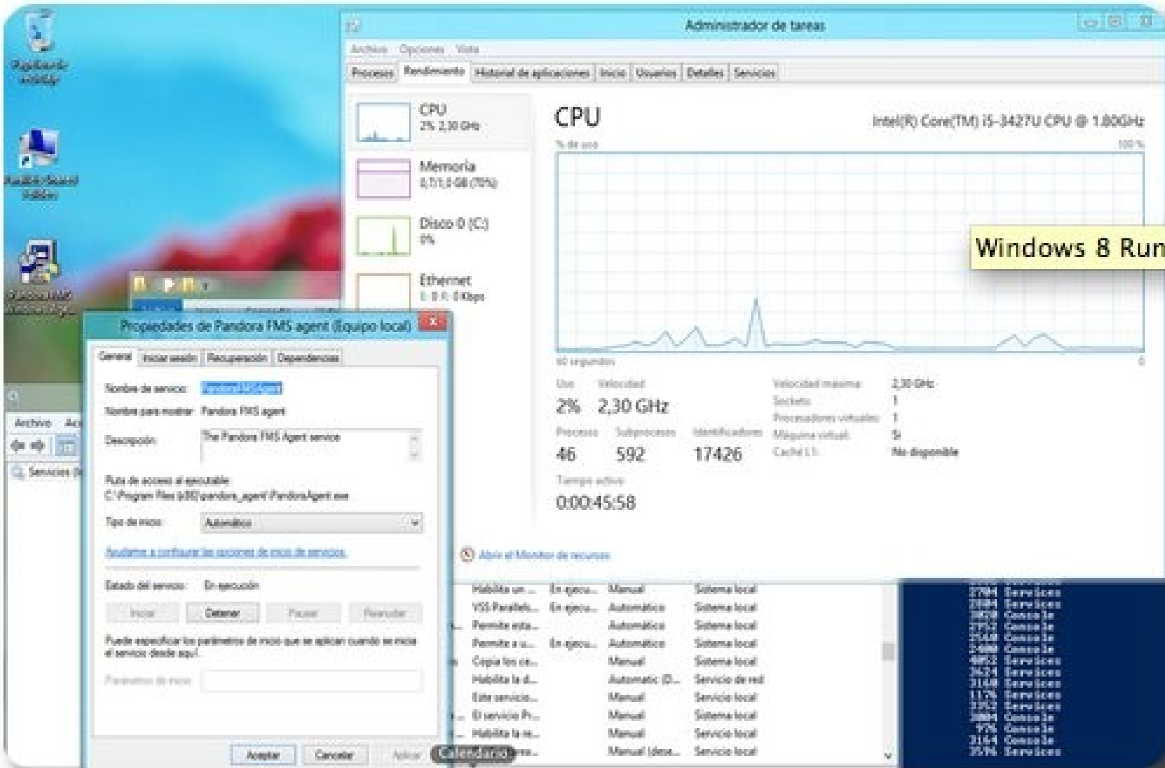

PANDORA FMS
All systems: Ready
Autorefresh
Events
ártica
Enter keywords to search
:: Operation ::
Tree view - Sort the agents by
Monitoring
Tactical view
Group view
Agent detail
Alert detail
Monitor detail
Services
Inventory
Recon view
SNMP console
Tree view
Modules groups
Agents/Modules view
Tree view
Network View
Reporting
View events
Workspace
Extensions
:: Administration ::
Manage monitoring
Monitor status
All
Search agent
Show
Agent name
NERV
IP Address
Interval
5 minutes
Description
Servidor de disco en red
Agent Version
4.0.2
Last contact / Remote
Next agent contact
Teamviewer ID
Agent access rate (24h)
NO DATA TO SHOW
PANDORAFMS
Go to agent detail
Servidores Internos (8 : 0 : 0 : 0)
beholder (15 : 15)
popeye (28 : 28)
nerv (3 : 3)
miles (13 : 13)
rembrandt (3 : 3)
carbon (26 : 26)
galaga (10 : 10)
Firewalls
Databases (0 : 0 : 0 : 0)
Network (3 : 0 : 0 : 0)
Pruebas
Workstations
Servicios (2 : 0 : 0 : 1)



Tree view - Sort the agents by
Monitor status All
Search agent
Show
Servers (0 : 0 : 0 : 1)
1My_Custom_Agent_name (3 : 3)
FreeDiskC 6
CPUUse 10
FreeMemory 47
Unknown (47 : 0 : 0 : 23)
Artica ST Architecture (16 : 0 : 0 : 1)
Web User Experience (6 : 0 : 0 : 0)
VMware Devices (14 : 0 : 0 : 4)
GIS Cars (11 : 0 : 0 : 0)
Energy Efficiency (11 : 0 : 0 : 0)
Agent name 1MY_CUSTOM_AGENT_NAME
IP Address 184.168.74.100
Interval 5 minutes
Description Created by firefly
Agent Version 4.0.1(Build 111220)
Position (Long, Lat) There is no GIS data.
Last contact / Remote
Next agent contact
Out of limits
Agent access rate (24h)



PANDORA FMS
[demo]
All systems: Ready
Autorefresh
Events
ártica
Enter keywords to search
Operation
Agent - _firefly
Monitoring
Tactical view
Group view
Agent detail
Alert detail
Monitor detail
Services
Inventory
Recon view
SNMP console
Tree view
Agents/Modules view
Modules groups
VMware View
Control keys...
Pandora FMS Remote Gateway



Autor

Equipo de Desarrollo de Pandora

http://www.pandorafms.com/



Willay News

GNU/LINUX | TECNOLOGÍA | CONOCIMIENTO

Novedades Tecnológicas

Bolígrafo Inteligente

Apareció un bolígrafo capaz de reconocer nuestros errores, y emite una leve vibración para advertirlo. Lernstift no sólo lo hará con los fallos caligráficos, sino que también corrige al equivocarse gramatical y ortográficamente.



El lápiz está programado para reconocer los movimientos asociados a cada letra y, en su modo caligrafía, es capaz de vibrar cuando una letra manuscrita quedó con una forma extraña o poco reconocible.

En su modo ortografía, es capaz de reconocer los errores ortográficos y gramaticales, y vibra una vez por un error ortográfico y dos veces por un error gramatical.

Ha sido diseñado por la empresa alemana Lernstift para ayudar a los jóvenes a aprender a escribir, pero podría hacerse popular con la gente de todas las edades. El lápiz, que se encuentra en la fase de prototipo, es capaz de recoger los errores en la formación de ortografía o letra en tiempo real.



El interior de LernStift incorpora sensores de movimientos y de presión, un módulo WiFi para transferir las transcripciones al ordenador, una batería y obviamente el software para detectar la escritura. Una de las cosas que más nos puede sorprender es que además de comportarse como un bolígrafo normal con su tinta para escribir sobre el papel también permite escribir en el aire. Al fin y al cabo los sensores detectan nuestros gestos en cualquier sitio, no importa donde los realicemos.

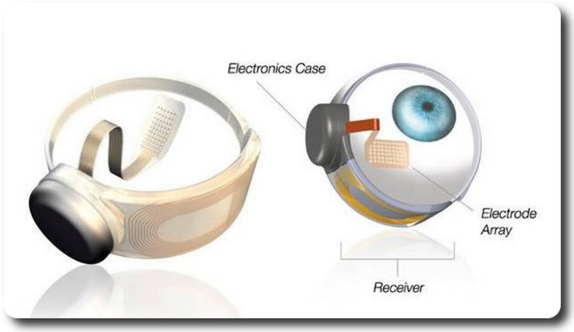


Falk y Wolsky Mandy, sus creadores, se inspiraron en los intentos de escritura temprana de su hijo, y no tardaron en comenzar a gestar este elemento que brindará muchas facilidades a jóvenes y adultos.

El LernStift es una utilidad bastante interesante para aquellos niños que estén empezando a escribir. La vibración es una forma de notificar que se ha cometido un error y obliga a reflexionar para encontrarlo. De momento, se puede reservar en su página web antes de que salga a la venta en agosto de este mismo año. Eso si, todavía se desconoce el precio oficial.

Ojo Biónico

Lo que hasta hace unas décadas era ciencia ficción hoy se convierte en realidad.



Luego de varios años de investigación, se ha logrado tener prótesis oculares verdaderamente sorprendentes, que ayudarán a inmuerables personas con deficiencias visuales y/o ciegos.



El dispositivo, bautizado como Argus 2, fue realizado por la empresa californiana Second Sight Medical Products y está compuesto por electródos implantados en la retina y unas lentes equipadas con una cámara en miniatura.



Con la salida de este dispositivo, se considera que a partir de ahora se espera la salidad de otros dispositivos "biónicos" que permitan a las personas con ciertas discapacidades volver a contar con el o los sentidos u órganos que perdieron funcionalidad o sufren de ciertas limitaciones.

Ahora comienza la era de los órganos biónicos.

Autor



Jenny Saavedra López
Diseño y Edición Revista Atix
jenny.saavedra@atixlibre.org

Libres para pensar, libres para decidir, libres para crear

Arte Libre

Te ofrecemos este espacio para mostrar tu Creatividad



Envíanos tus diseños y creaciones para publicarlos


atix
Libre

Contacto

Para solicitar cualquier información, puedes contactar a:

- Esteban Saavedra López (esteban.saavedra@atixlibre.org)
- Jenny Saavedra (jenny.saavedra@atixlibre.org)

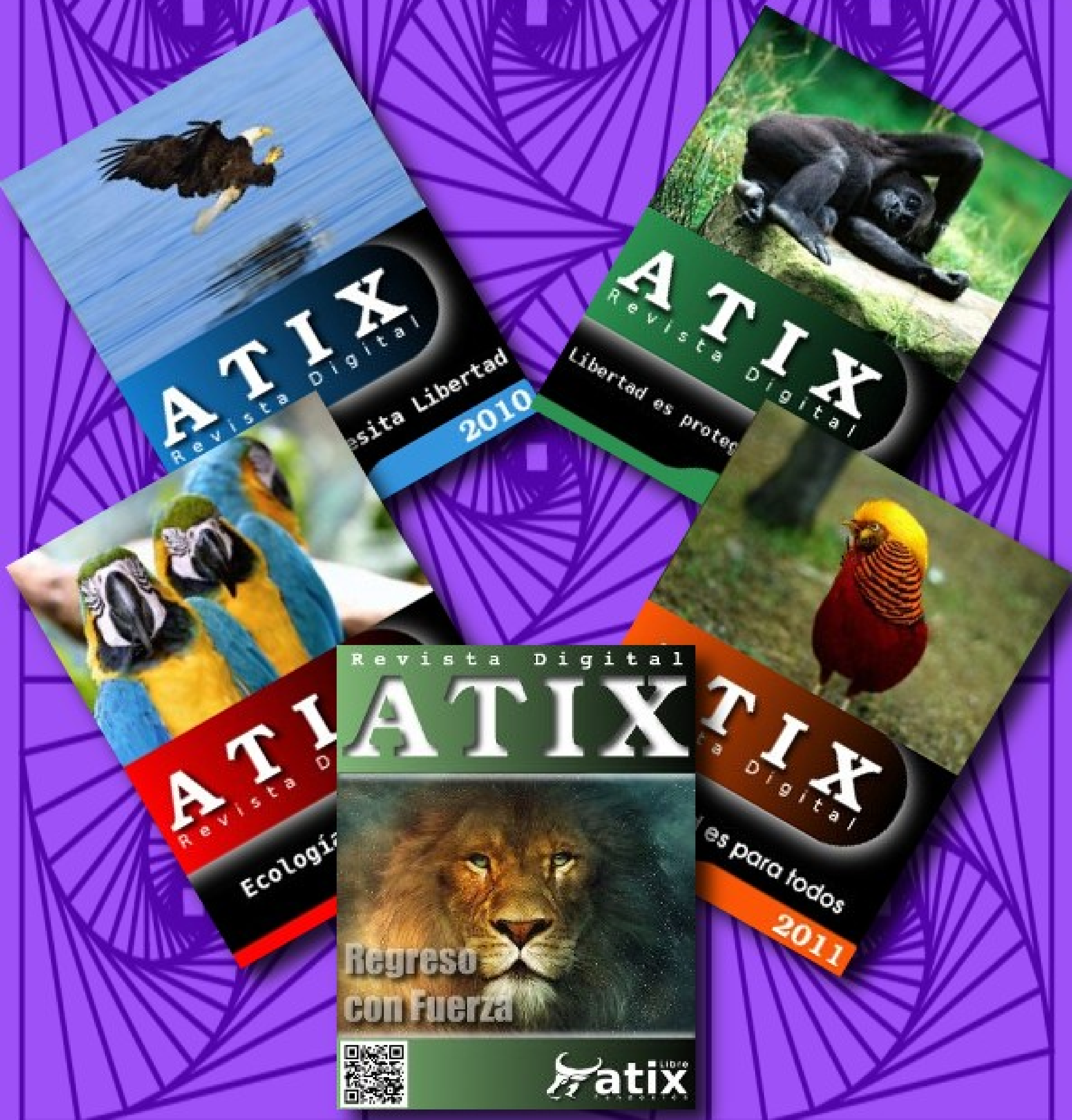
Visita nuestro sitio web y descarga todos los números

- http://revista.atixlibre.org

Envío de Artículos

Te invitamos a participar de la Revista Atix enviándonos artículos referidos a las siguientes áreas :

- Instalación y personalización de aplicaciones
- Scripting
- Diseño gráfico
- Programación y desarrollo de aplicaciones
- Administración de servidores
- Seguridad
- y cualquier tema enmarcado dentro del uso de Software Libre
- Conocimiento Libre
- Tecnología Libre
- Cultura Libre
- Trucos y recetas.
- Noticias.


ATIX
Revista Digital
esita Libertad
2010
ATIX
Revista Digital
Libertad es proteg
Revista Digital
ATIX
Regreso
con Fuerza
atix
Ecología
TIX
es para todos
2011

Libre
atix
Fundación

Por un Mundo Ético, Libre y Justo

http://revista.atixlibre.org